

# MLP\_sprawozdanie\_Olaf\_Werner\_291139

April 18, 2020

## 1 Wstęp

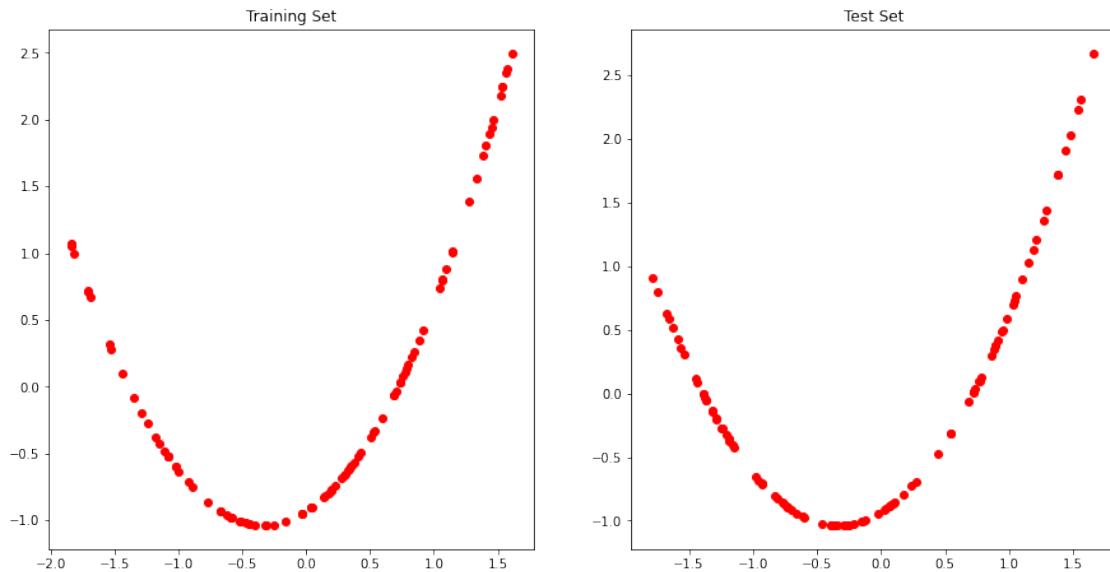
Celem tego sprawozdania jest przeprowadzenie testów mojej implementacji sieci neuronowej. Będzie parę eksperymentów. Każdy eksperiment bedzie opisany przez warunki eksperymentu, hipotezę, wyniki i wnioski. Z założenia wszystkie dane na których przeprowadzam eksperymenty są zestandardyzowane. Poza tym dla problemów klasyfikacji dane są w kodowaniu "one-hot". Podczas eksperymentów sieci będą startować z tymi samymi wagami jeśli mają tyle samo neuronów w każdej warstwie dla danego zbioru. Wagi sieci będą inicjowane losowo, jednostajnie z przedziału [-1, 1]. Dla problemów klasyfikacji używany jest softmax na ostatniej warstwie. Dla problemów klasyfikacji liczymy accuracy i cross entropy, ale trenujemy używając cross entropy. Dla problemów regresji używamy MSE.

## 2 Eksperyment 1 wpływ ilości warstw i neuronów na uczenie

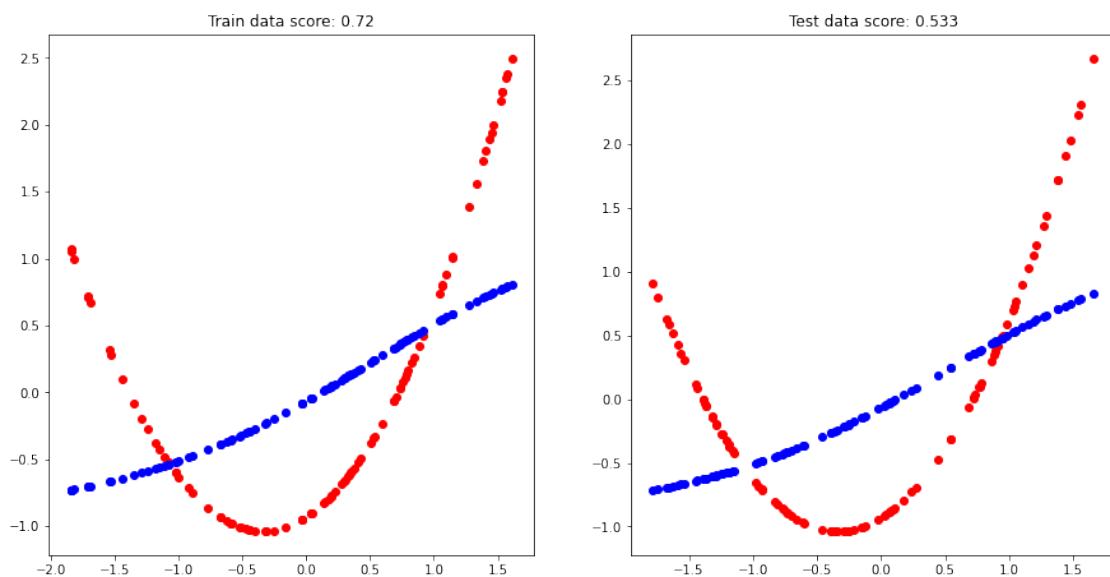
Zbadamy wpływ ilości warstw i neuronów na proces uczenia. Zastosujemy tradycyjną metodę spadku gradientu, za learning rate biorąc 0.1, a naszą funkcją aktywacji będzie sigmoid. Za stop mamy przekroczenie liczby epok równe 300 lub bezwzględna zmianę błędu mniejszą niż  $10^{**-6}$ . Nie używamy batchy. Takie warunki będziemy nazywać "podstawowymi". Będziemy testować następujące architektury: [5],[5,5],[10],[3,4,3],[6,6,6],[3,3,3,3]. Będziemy przeprowadzać testy na zbiorach: xor3, rings5-regular, square-simple i multimodal-small

**3 Hipoteza: im więcej połączeń między neuronami tym lepiej, zaś sama liczba neuronów nie jest już tak znacząca**

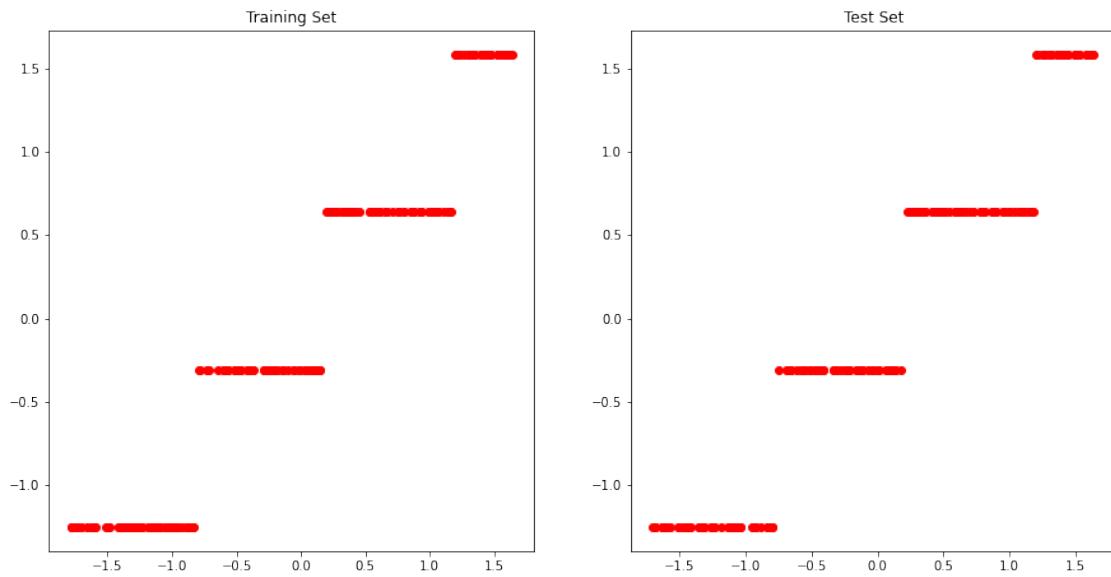
### 3.1 square-simple



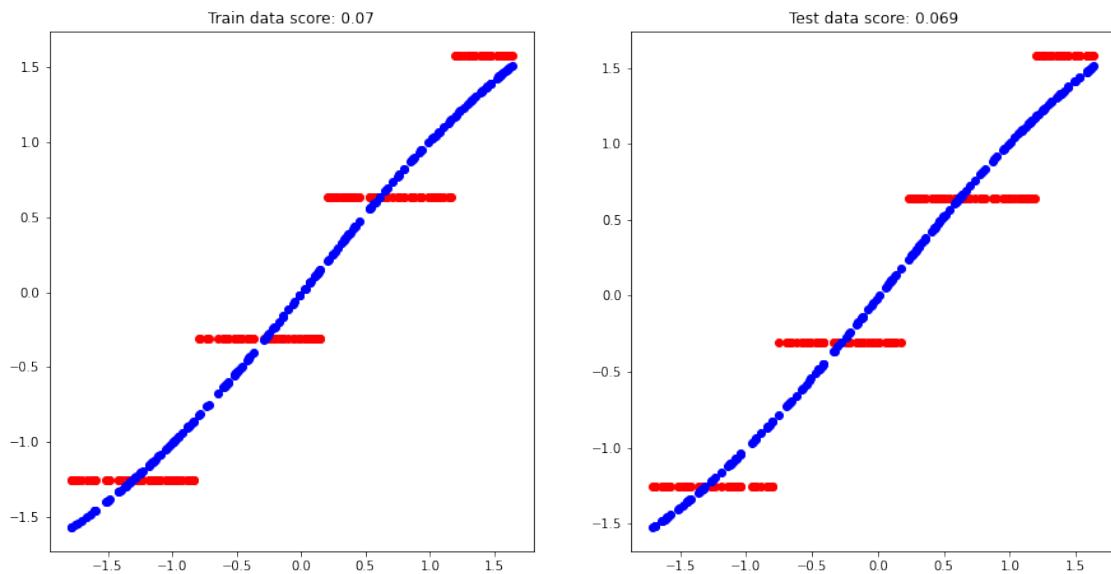
### 3.2 Najlepszy model



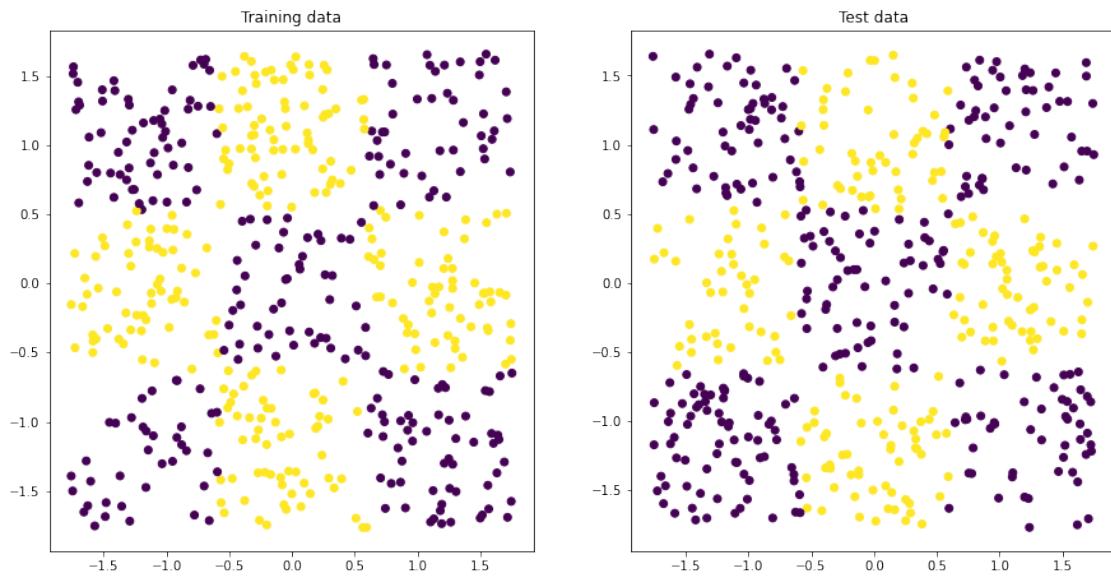
### 3.3 multimodal-small



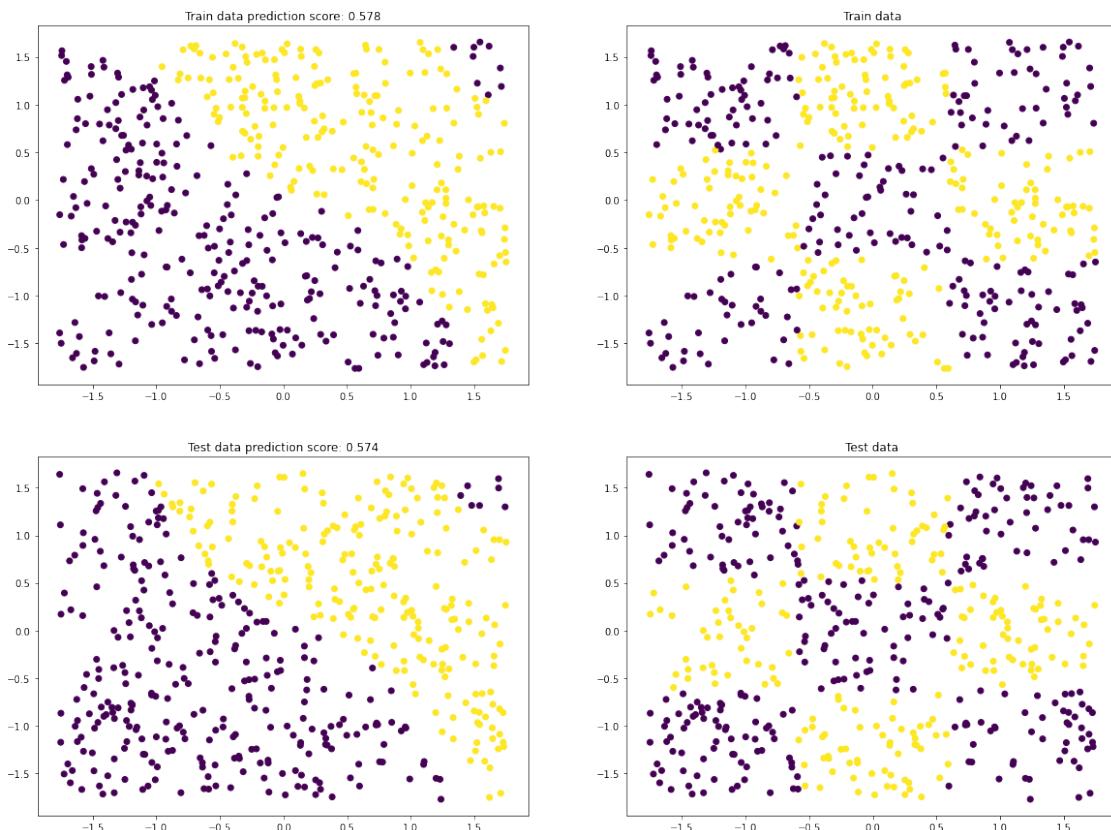
### 3.4 Najlepszy model



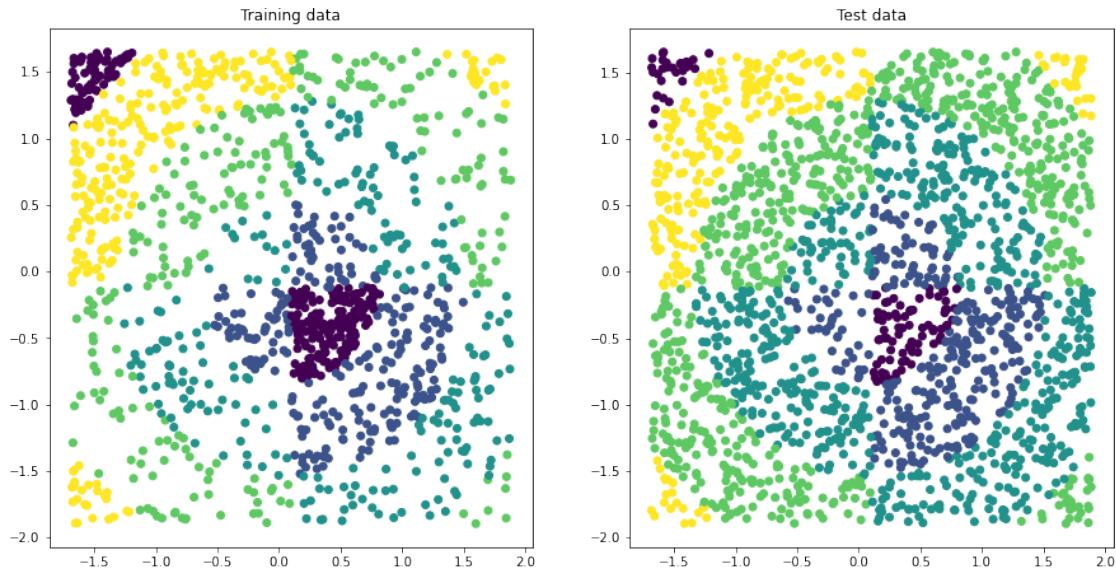
### 3.5 xor3



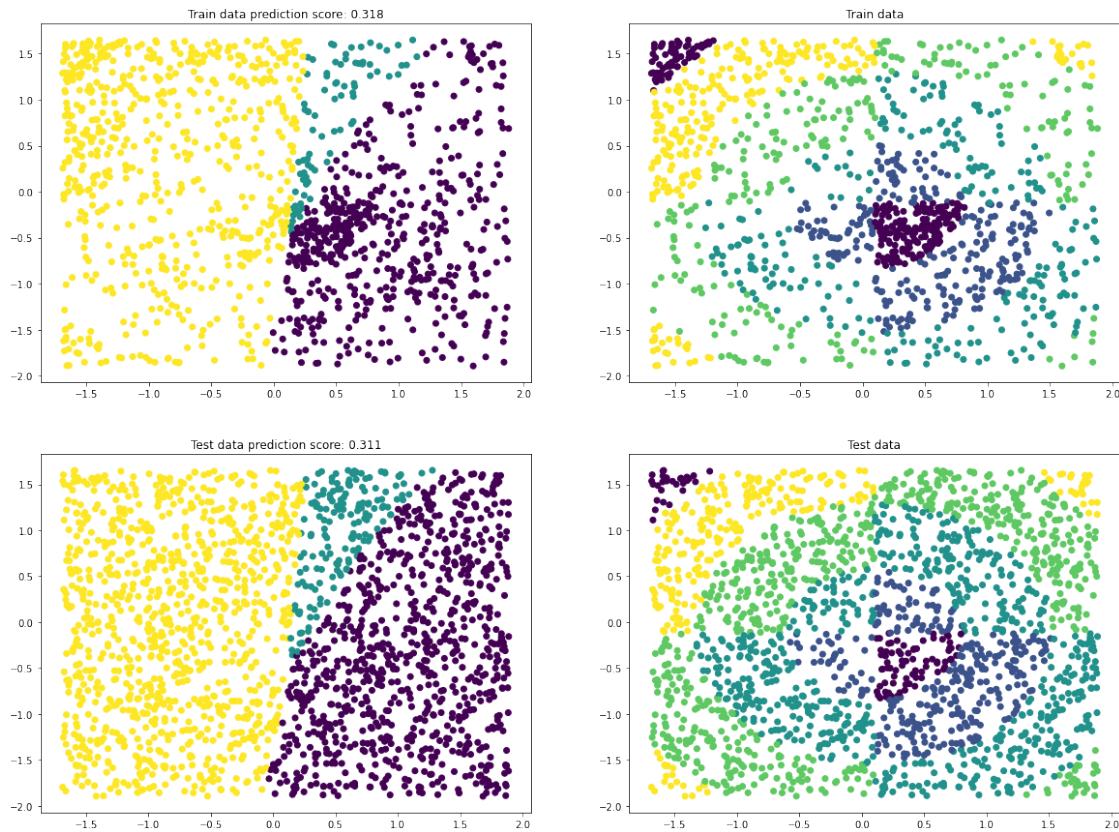
### 3.6 Najlepszy model



### 3.7 rings5-regular

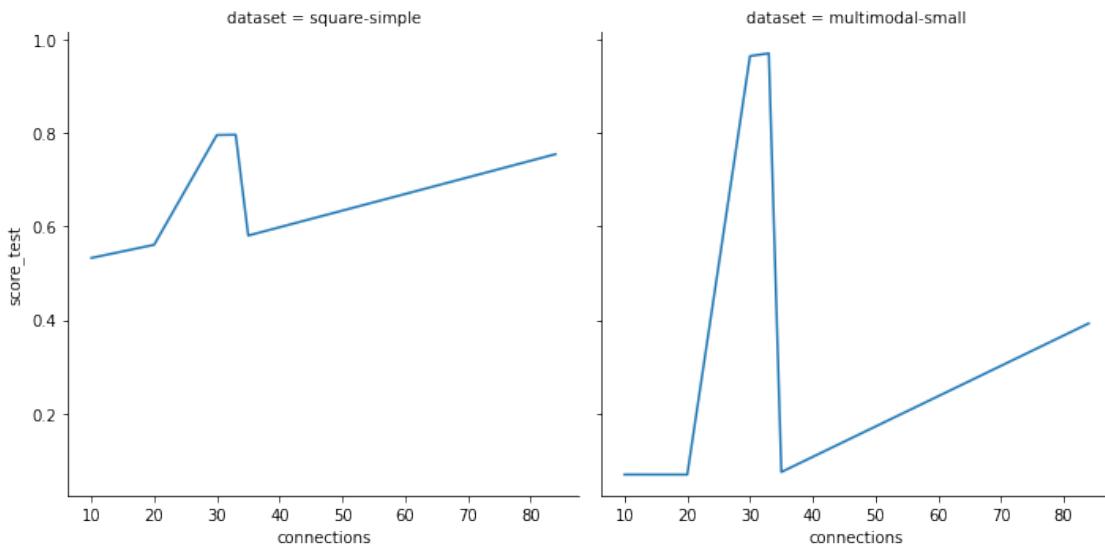


### 3.8 Najlepszy model

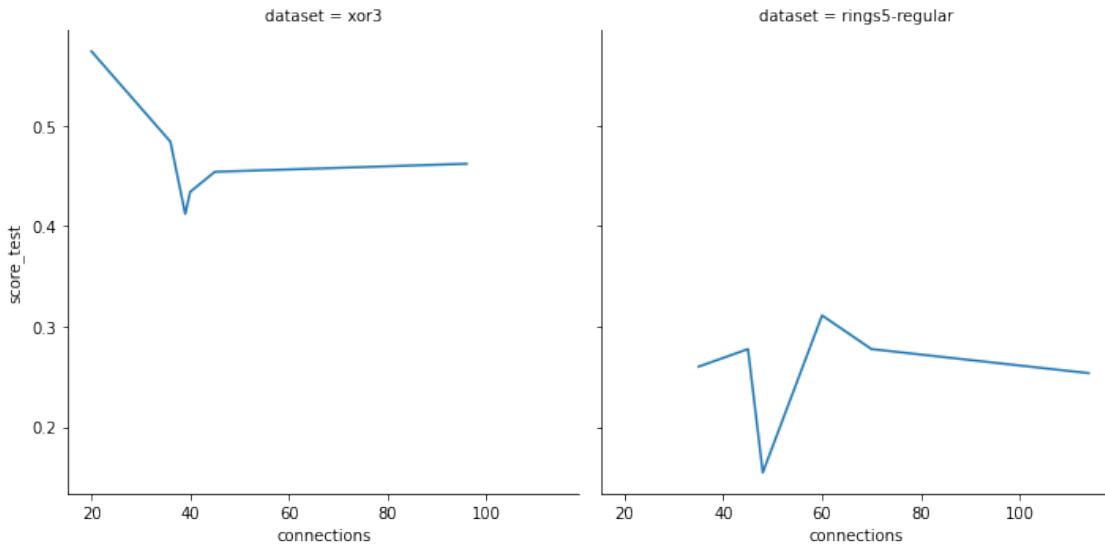


## 3.9 Wykres wyniku od liczby wag

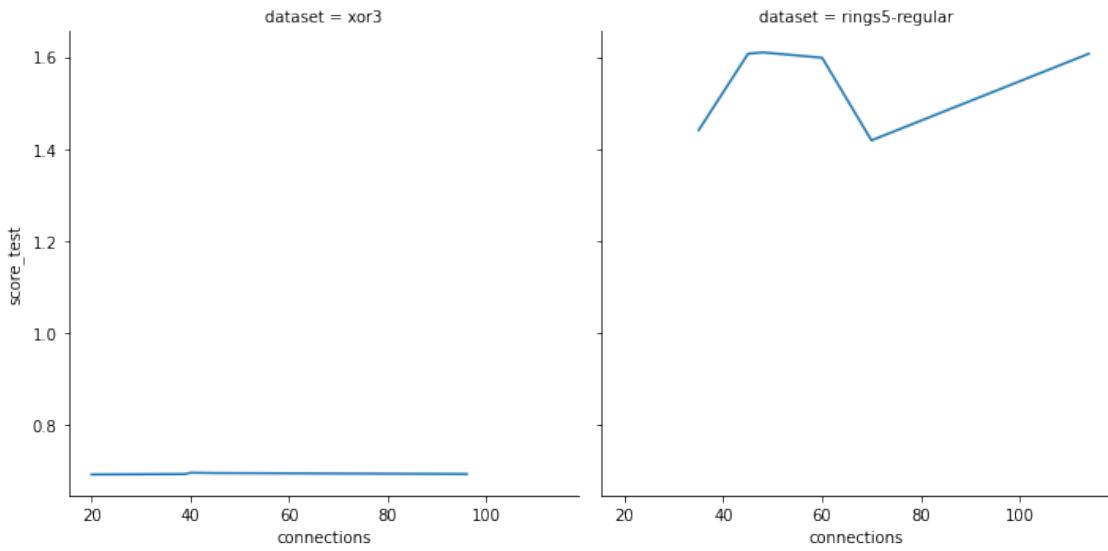
### 3.9.1 MSE



### 3.9.2 Accuracy



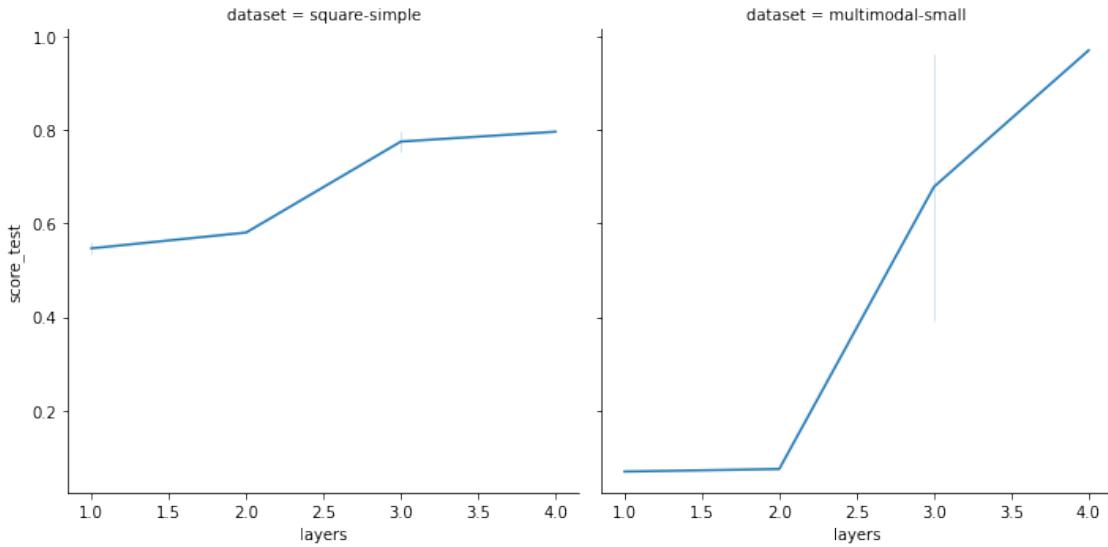
### 3.9.3 Entropy



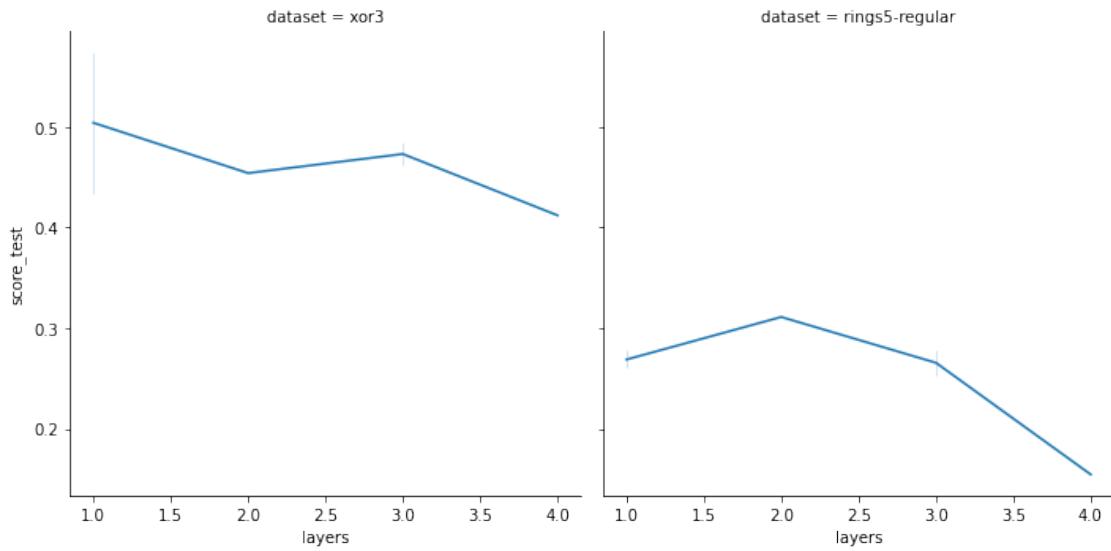
Ogólnie to mamy obserwację odstającą jest to architektura [3,3,3,3] najwyraźniej architektura która jest dłuża, ale wąska jest złą architekturą. Po za tym to ilość wag nie ma wpływu tu oczywistego wpływu

## 3.10 Wykres wyniku od liczby warstw

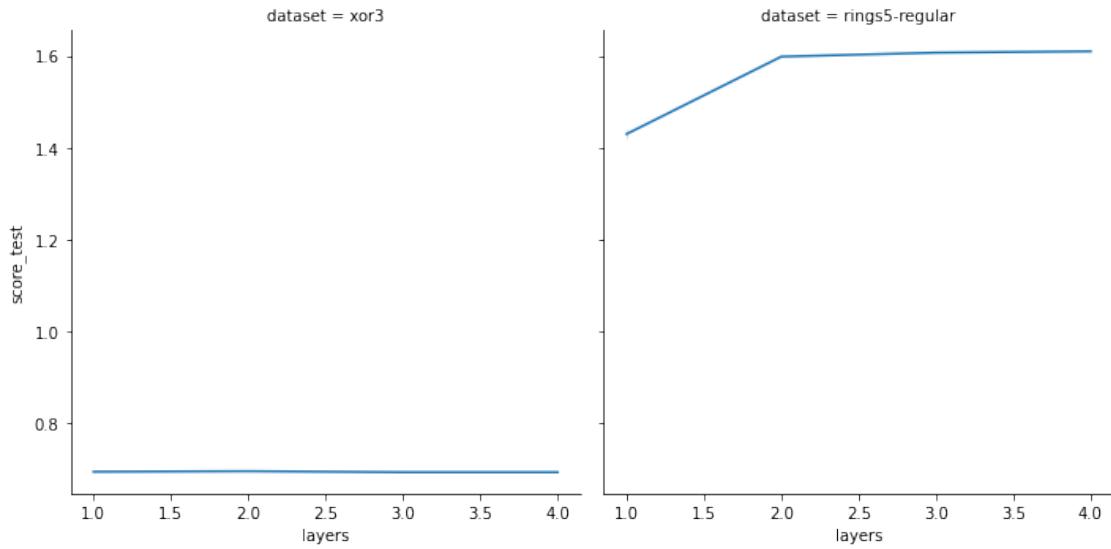
### 3.10.1 MSE



### 3.10.2 Accuracy



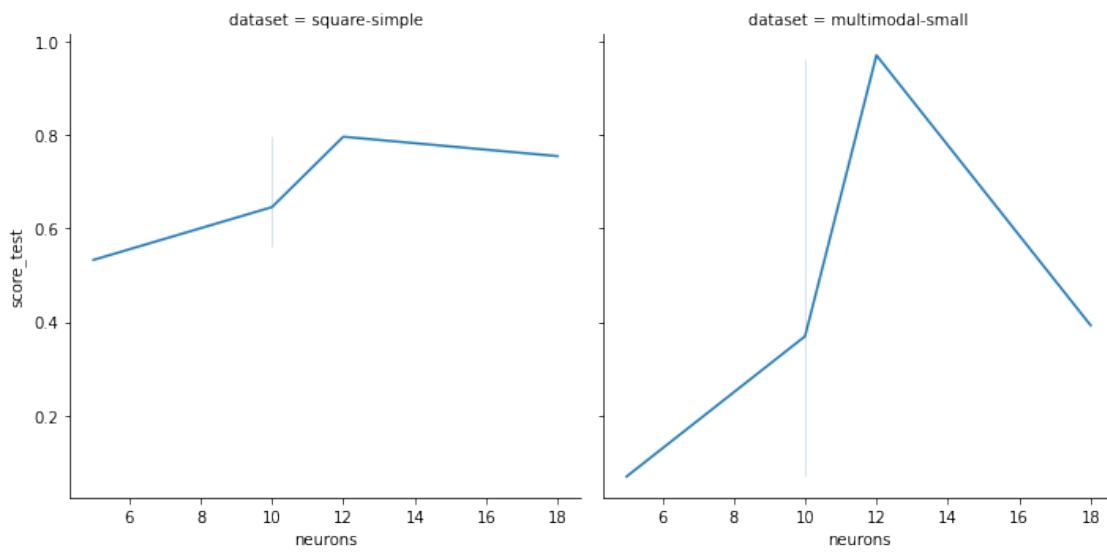
### 3.10.3 Entropy



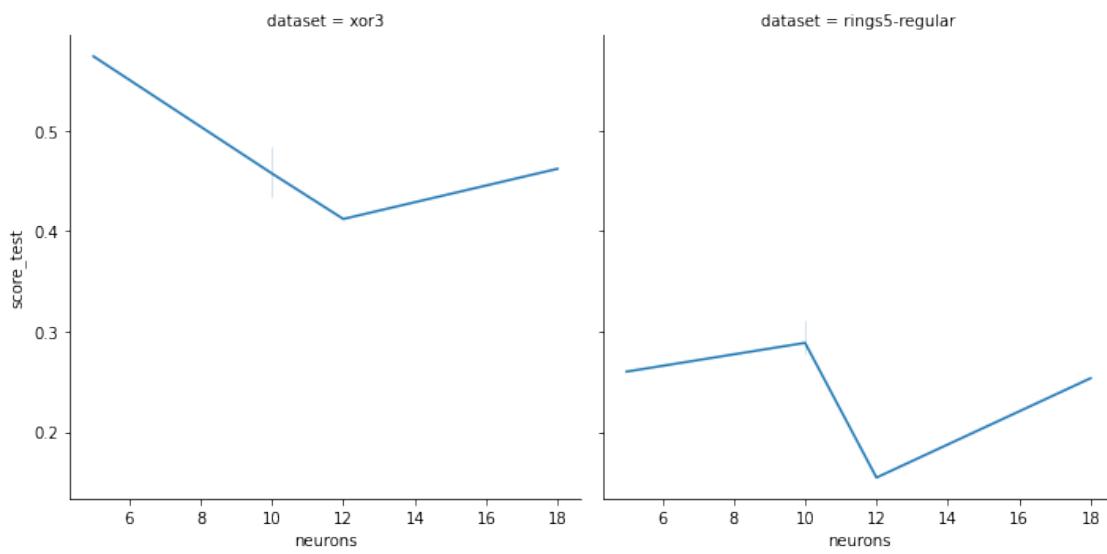
Ponownie architektura [3,3,3,3] jest złą architekturą. Po za tym to ilość warstw wydaje się mieć ujemny wpływ tzn im więcej warstw tym gorzej

## 3.11 Wykres wyniku od liczby neuronów

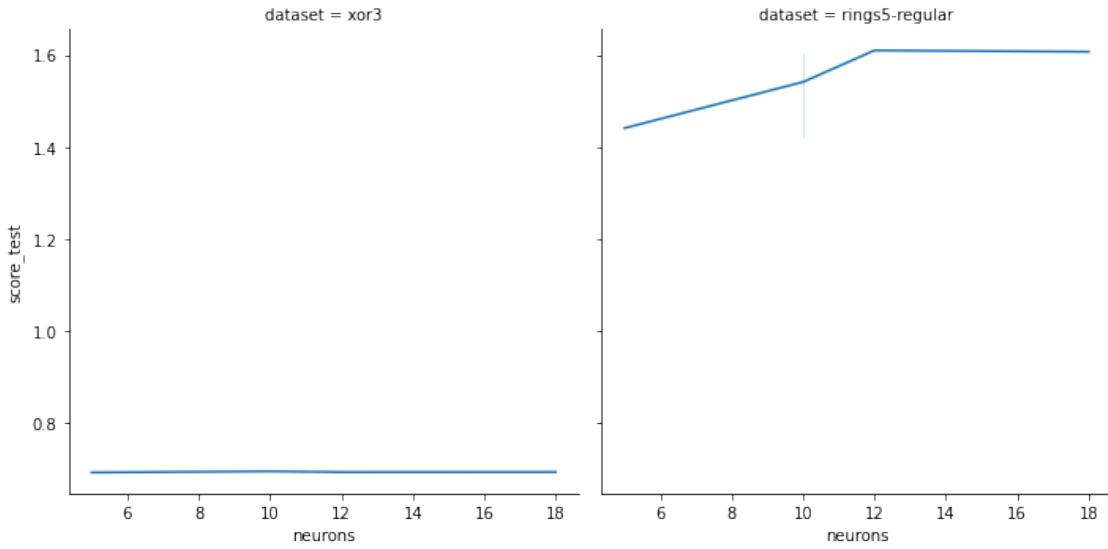
### 3.11.1 MSE



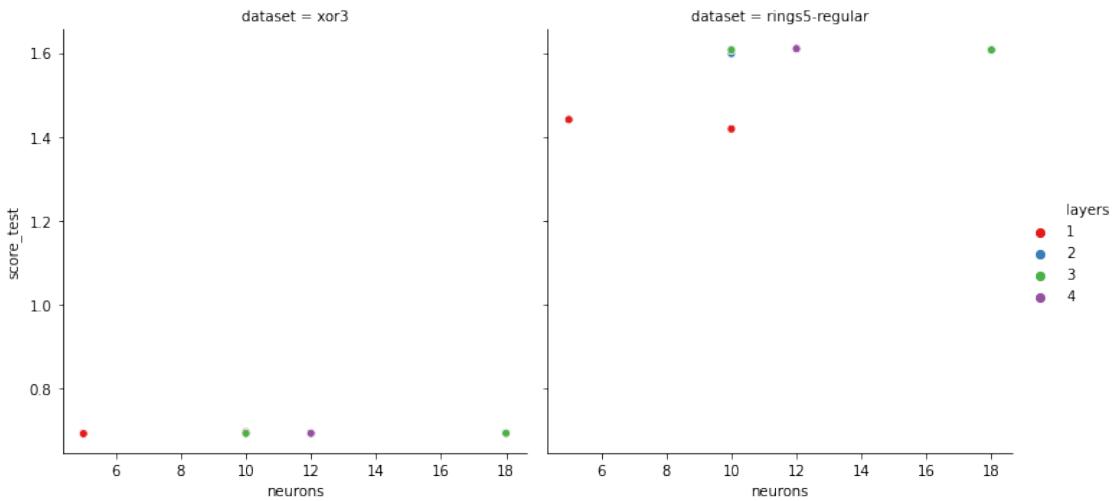
### 3.11.2 Accuracy



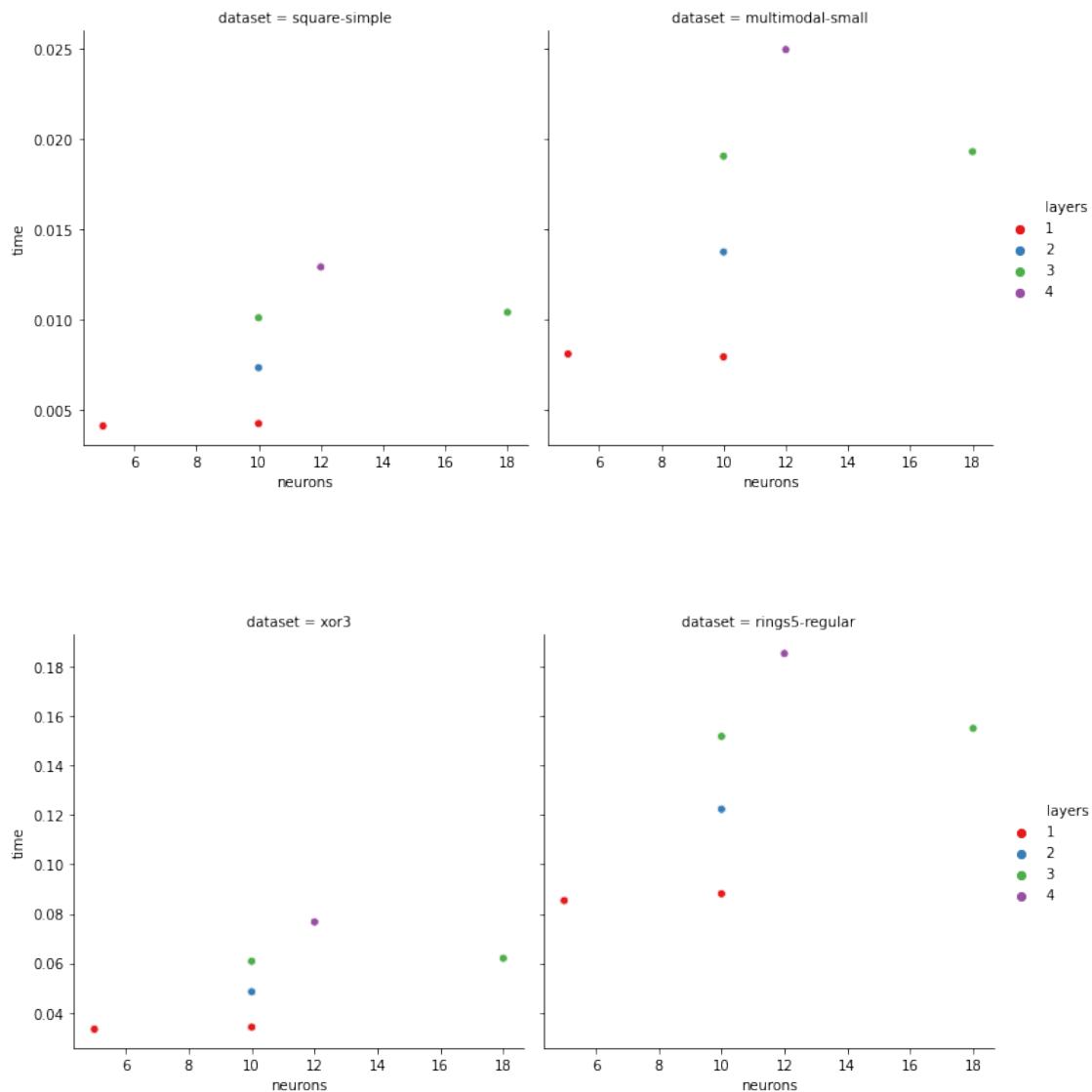
### 3.11.3 Entropy



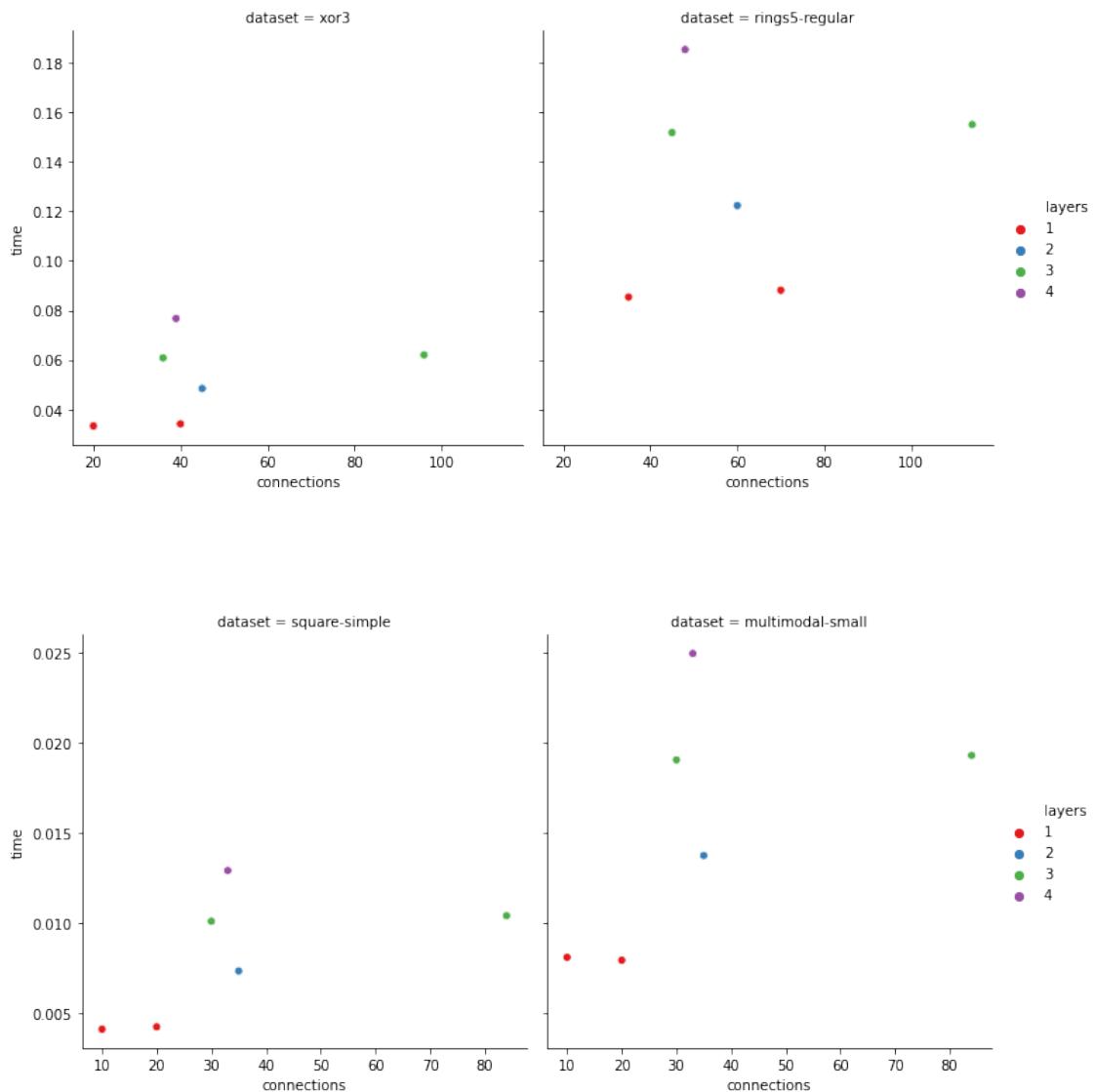
Ponownie [3,3,3,3] jest słaba, ale co ciekawe [10] daje lepsze wyniki niż [5,5] oraz [3,4,3]. Możliwe że szerokość sieci jest ważna.



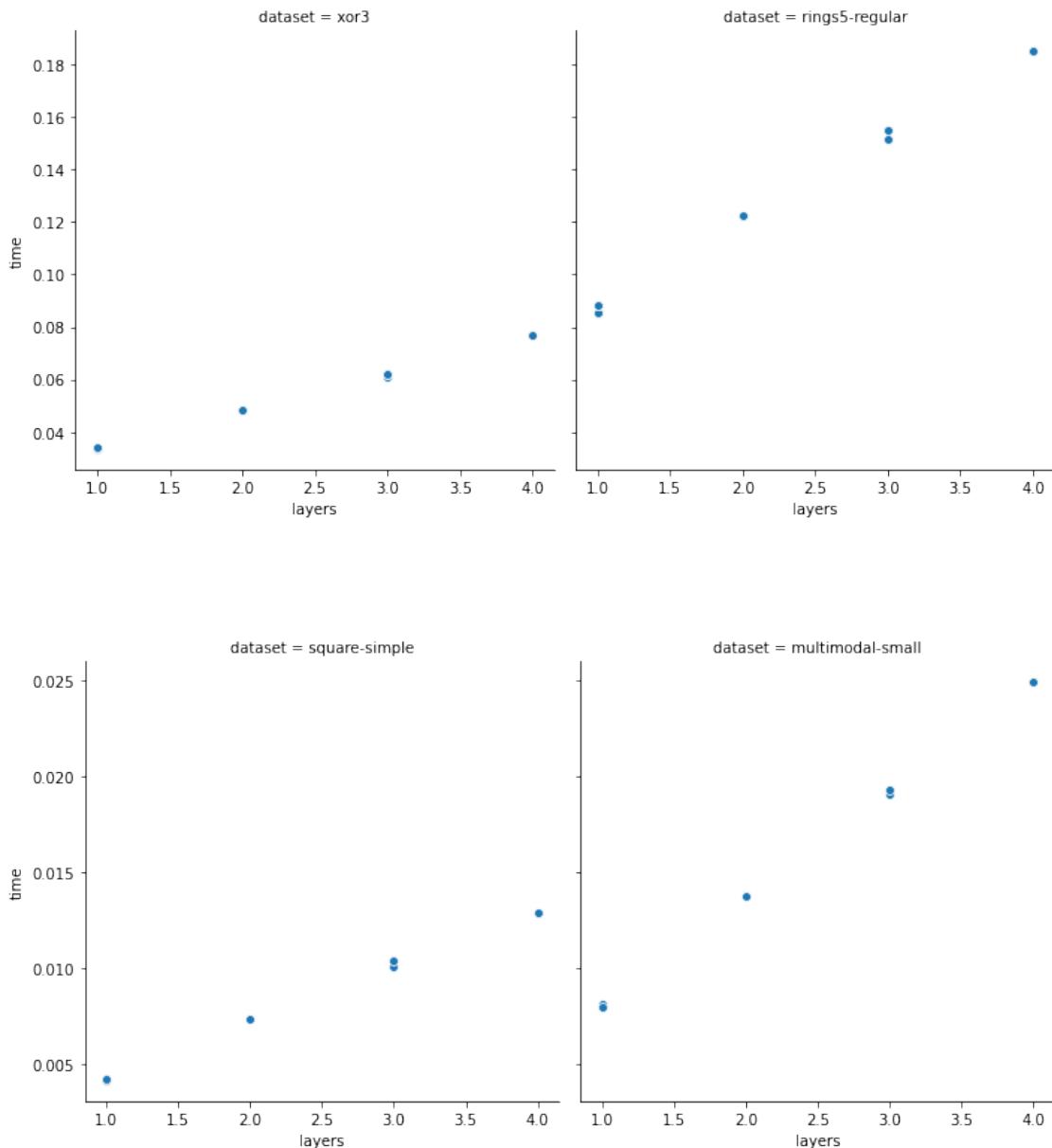
### 3.12 Wykres czasu od liczby neuronów i warstw



### 3.13 Wykres czasu od liczby wag i warstw

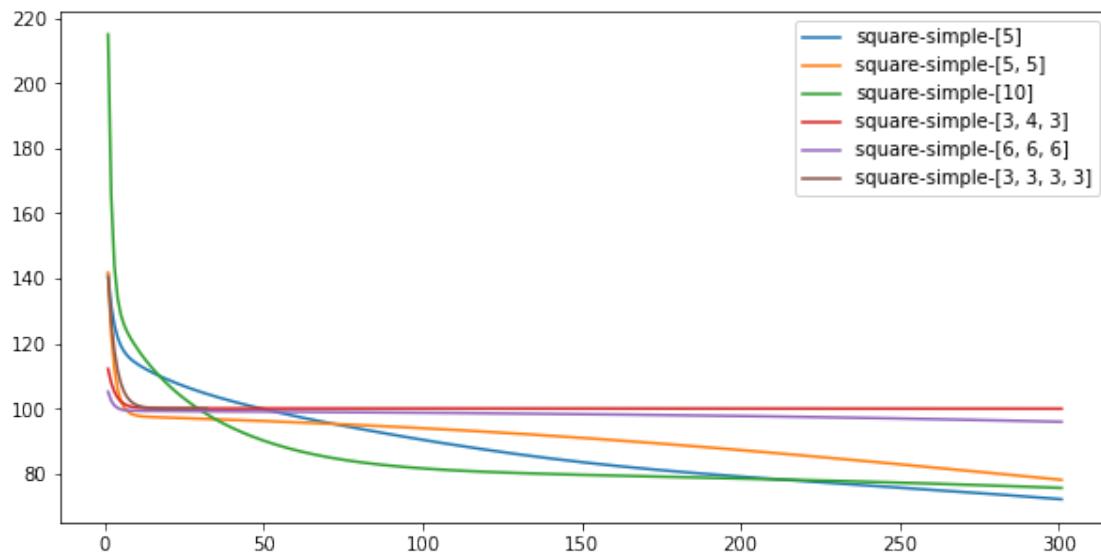


### 3.14 Wykres czasu od liczby warstw



Wniosek jest taki że im więcej warstw tym dłuższy średni czas na epoke, liczba neuronów w warstwie nie jest tak kluczowa.

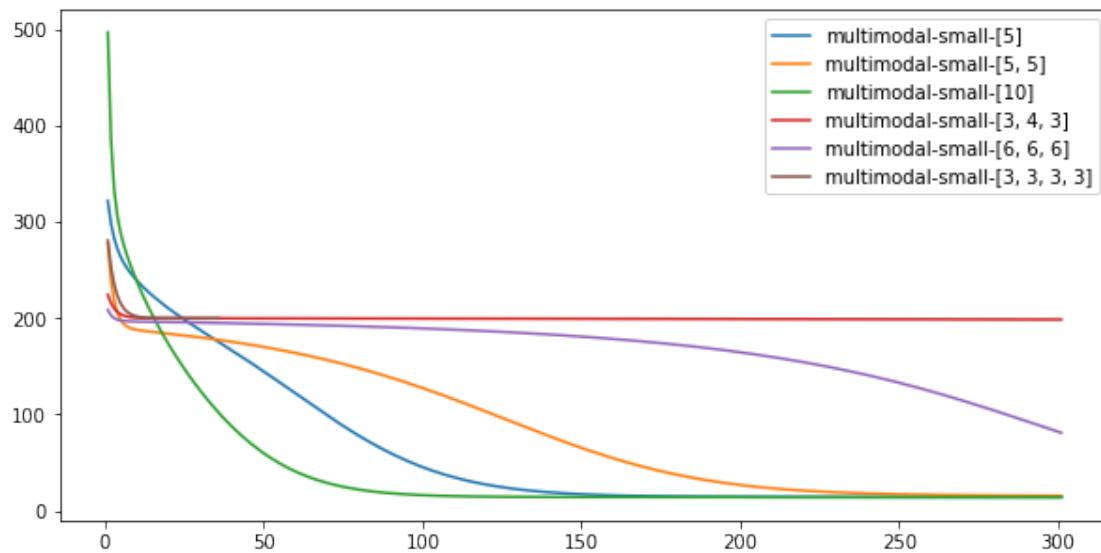
### 3.15 Wykres błędów square-simple



Widzimy dominację szerokich sieci nad wąskimi

### 3.16 Wykres błędów multimodal-small

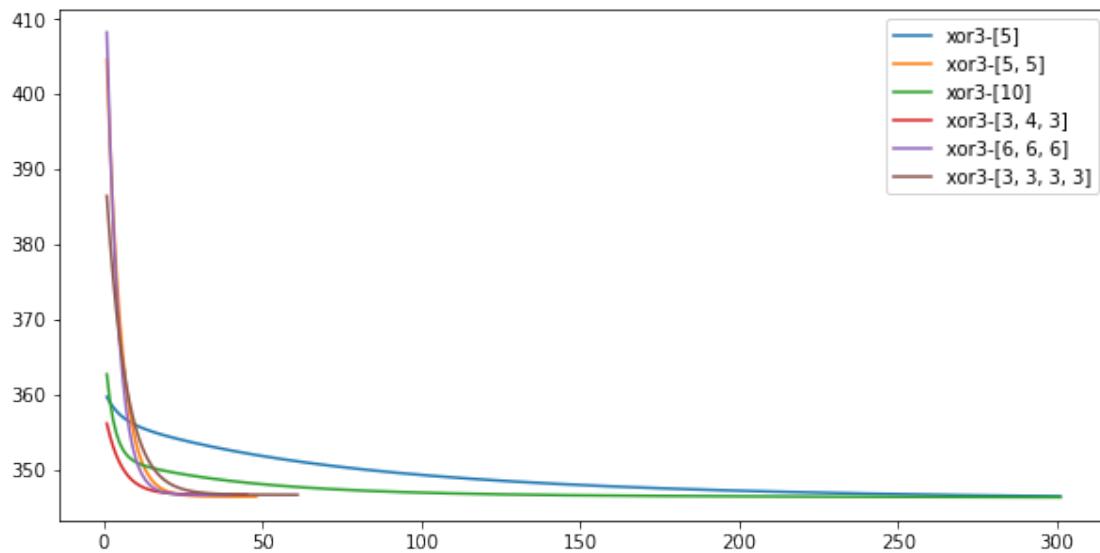
[1067]: <matplotlib.legend.Legend at 0x7f9eb6fa7cf8>



Tutaj można nie dojrzeć, ale [3,3,3,3] zakończyła uczenie przedwcześnie, zaś [10] jest najlepsza, im więcej warstw tym gorzej!

### 3.17 Wykres błędów xor3

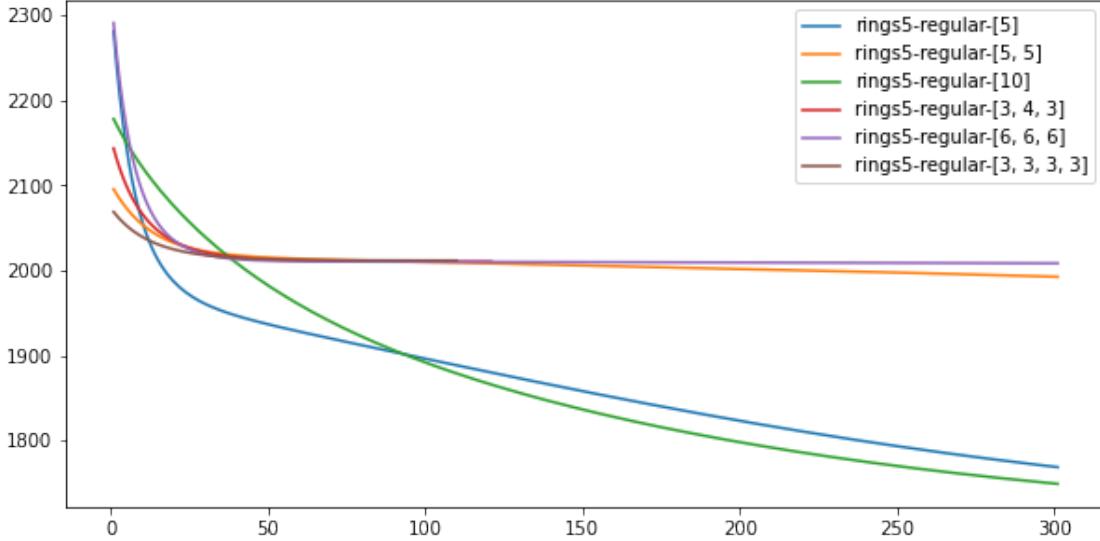
[1068]: <matplotlib.legend.Legend at 0x7f9eb6fb0fd0>



Tutaj jest mniej oczywiste, ale [10] jest znakomitą, ale nie tak dominującą architekturą tu też widać że jest lepsza od [5] i [5,5]

### 3.18 Wykres błędów rings5-regular

[1069]: <matplotlib.legend.Legend at 0x7f9eb6e90fd0>



Ponownie [10] jest najlepsze, zaś wąskie architektury wcześniej zakończyły uczenie

## 4 Podsumowanie

Przede wszystkim pokazaliśmy że szerokość warstwy jest bardzo ważna, architektury które są wąskie i długie, są gorsze niż te które są szerokie i krótkie. Ilość warstw okazała się najważniejsza przy czasie trwania procesu uczenia na epokę. Należy pamiętać że architektura [10] miała największą pierwszą i ostatnią warstwę, więc możliwe że to jest ważne.

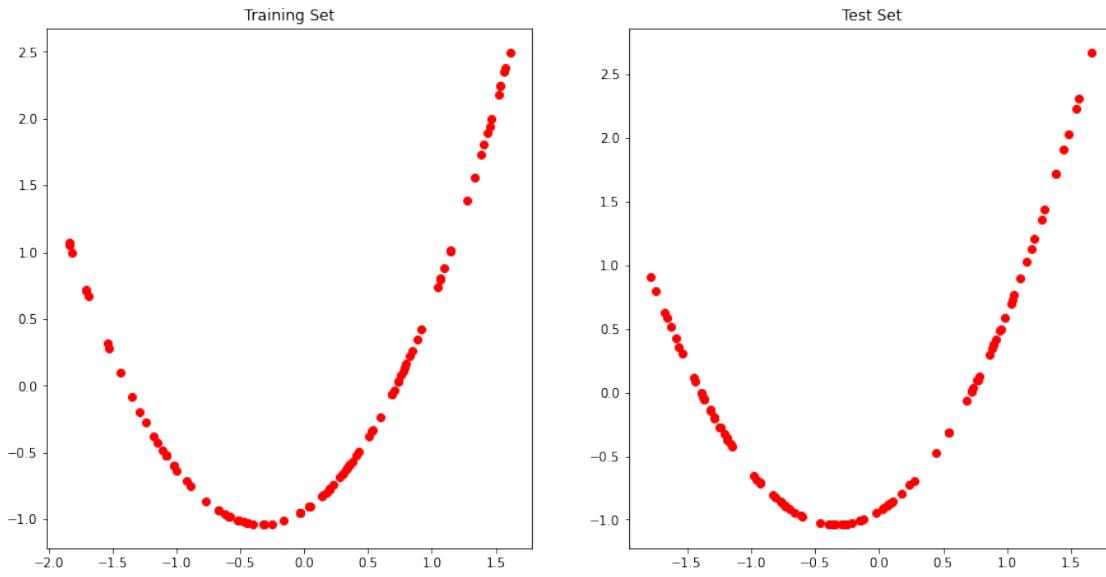
## 5 Eksperyment 2 wpływ batchy na uczenie

Spróbujemy przyspieszyć proces uczenia poprzez używanie tzw. batchy czyli podczas wyznaczania gradientu będziemy brali tylko część zbioru danych. Rozpatrzymy branie całego zbioru, 10% i 20%. Przetestujemy na zbiorach square-simple, steps-small i multimodal-small. Zbiory danych uczących są mieszane na samym początku, przed jakimkolwiek uczeniem. Będziemy testować na architekturze [8,8]. Będziemy używać zbiorów square-simple, steps-small, multimodal-small.

## 6 Hipoteza: branie części zbioru danych znacznie przyspieszy uczenie, ale może pogorszyć wyniki

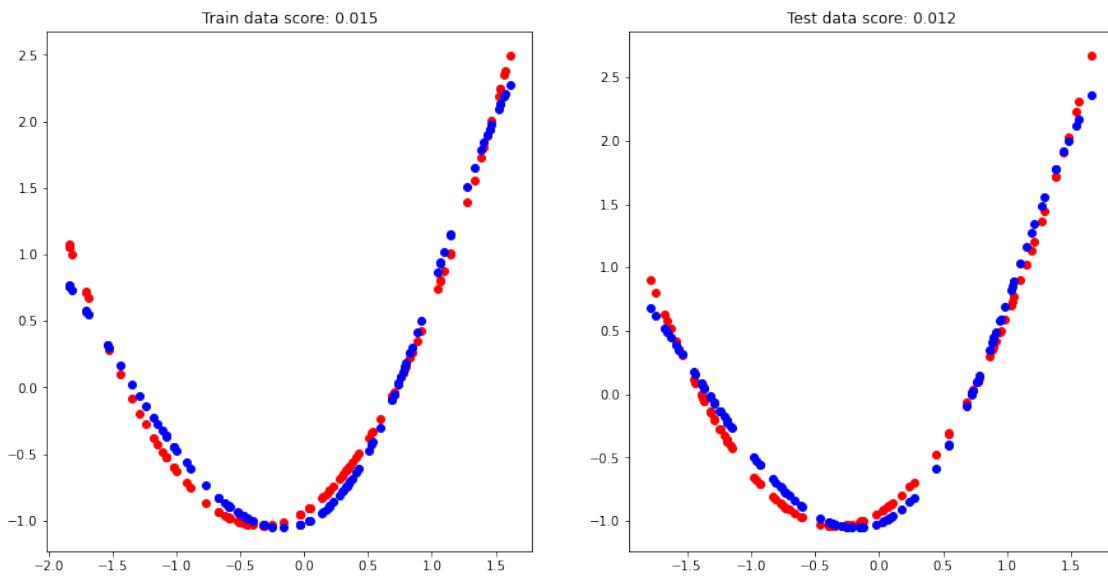
### 6.1 square-simple

[1072]: [Text(0.5, 1.0, 'Test Set')]



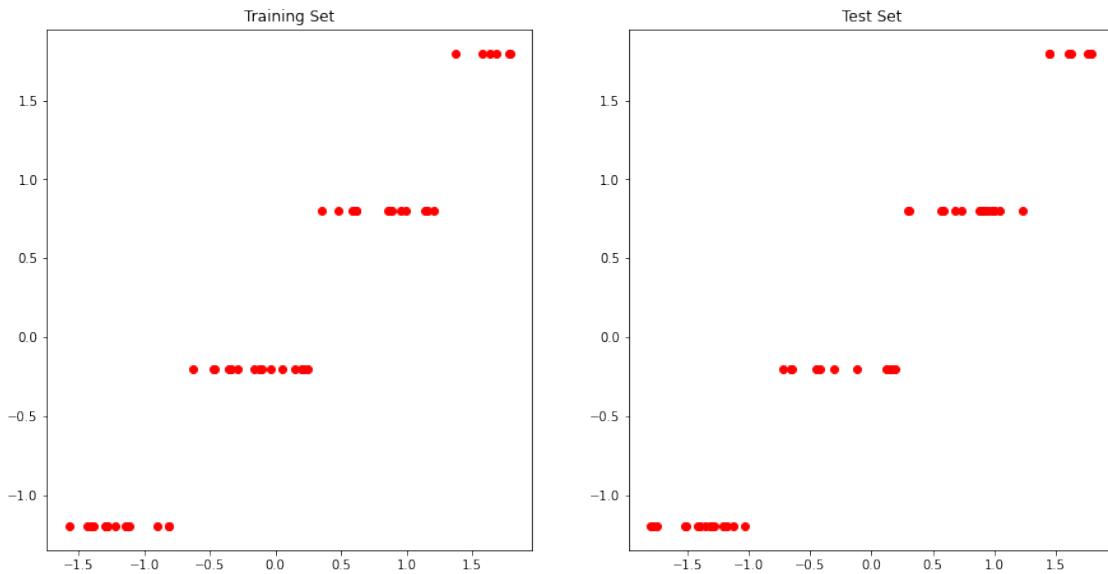
### 6.2 Najlepszy model

[1074]: [Text(0.5, 1.0, 'Test data score: 0.012')]

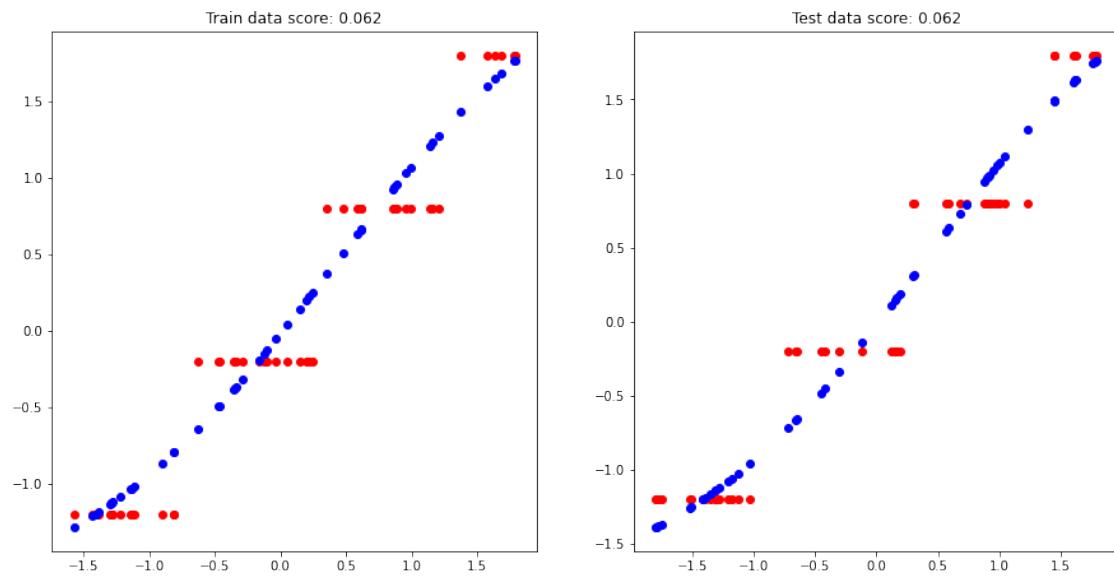


### 6.3 steps-small

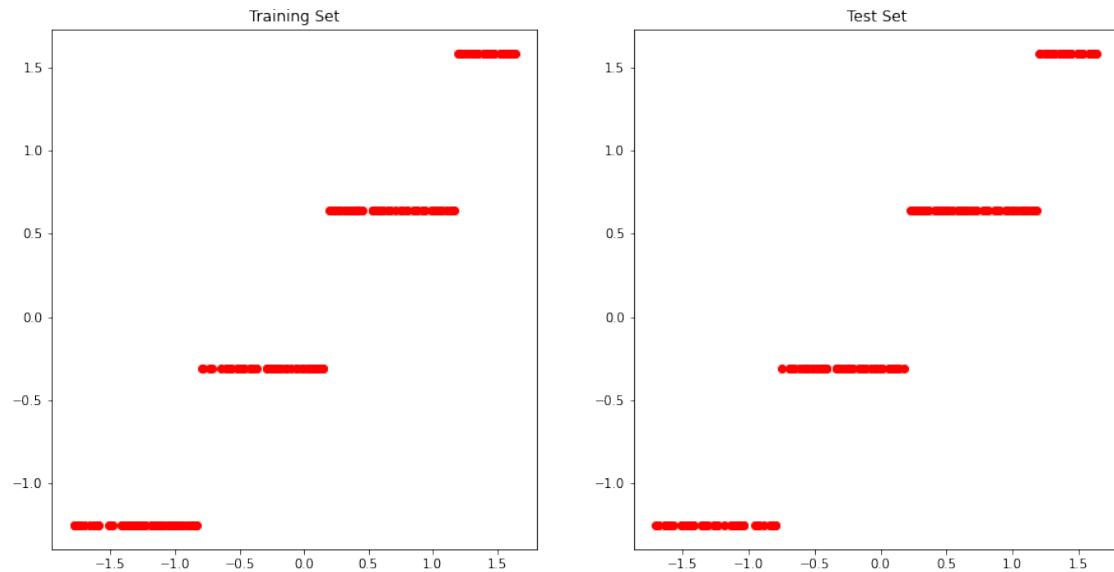
[1077]: [Text(0.5, 1.0, 'Test Set')]



## 6.4 Najlepszy model

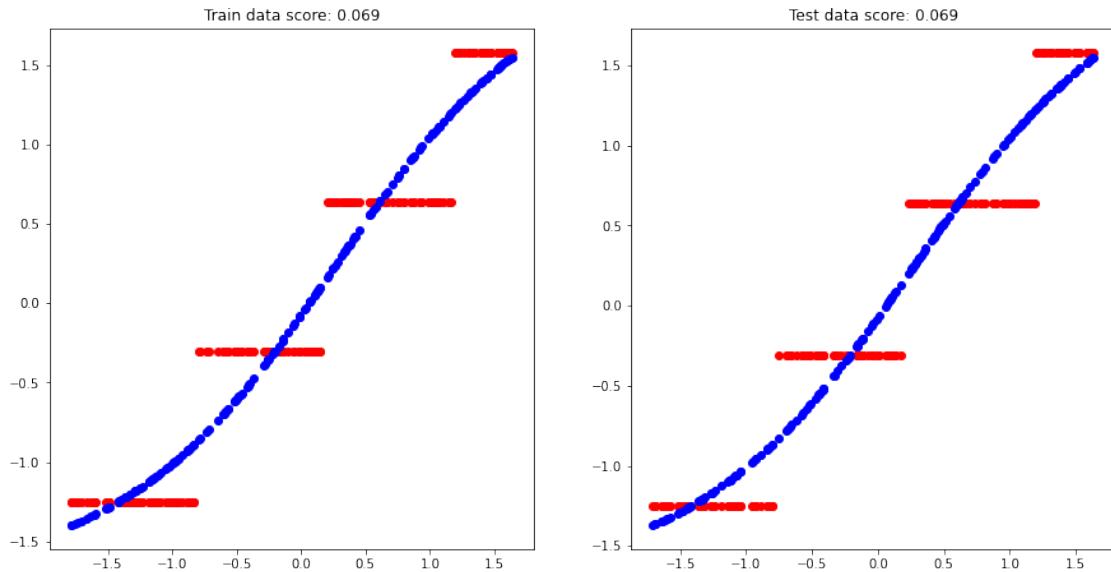


## 6.5 multimodal-small



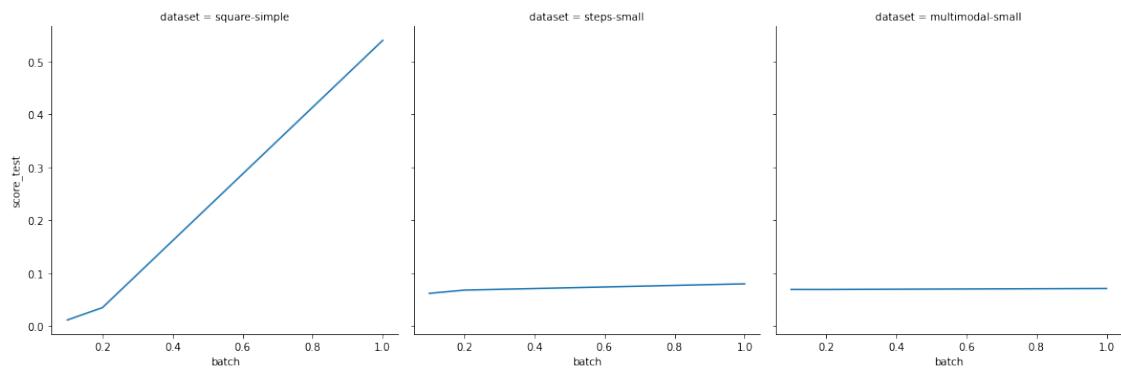
## 6.6 Najlepszy model

[1084]: [Text(0.5, 1.0, 'Test data score: 0.069')]



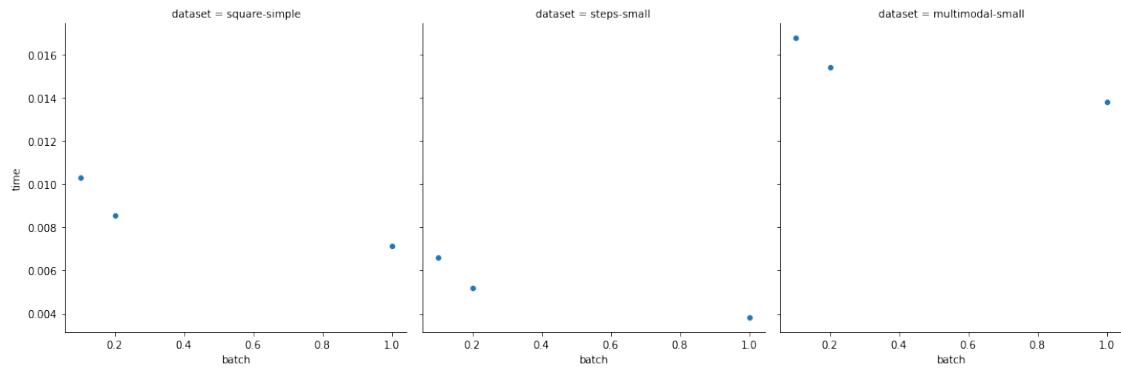
## 6.7 Wykres wyniku od batch

### 6.7.1 MSE



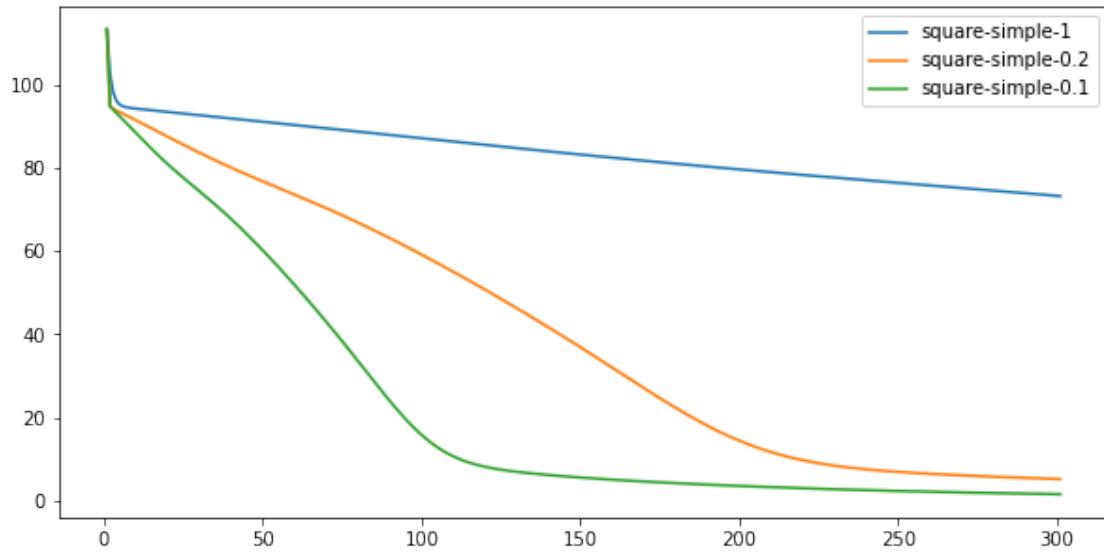
10% batch daje najlepsze rezultaty.

## 6.8 Wykres czasu od liczby batch



Batche mają wpływ na tempo uczenia, ale im większy zbiór tym mniejszy

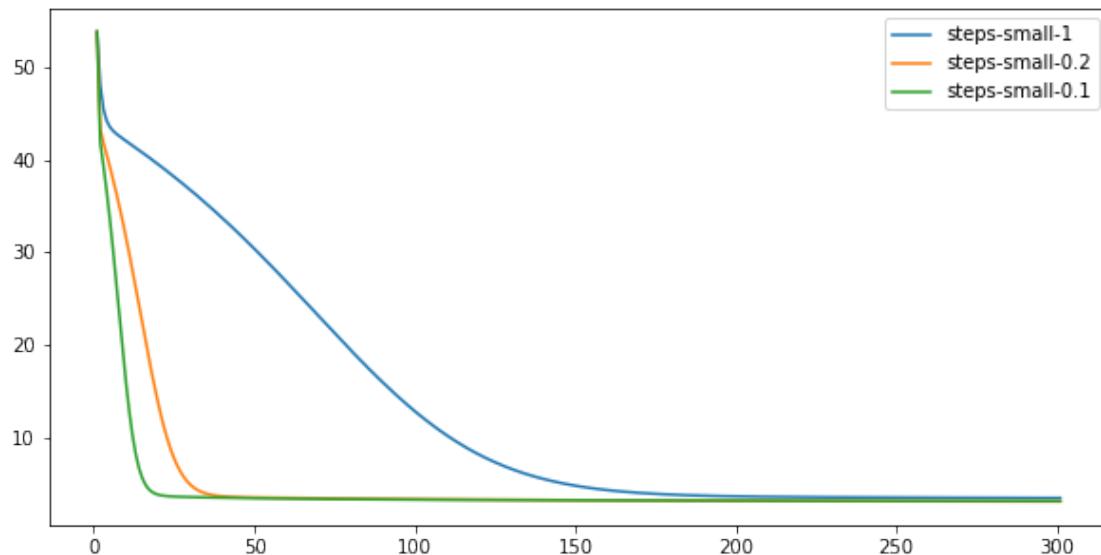
## 6.9 Wykres błędów square-simple



10% batch najlepszy

## 6.10 Wykres błędów steps-small

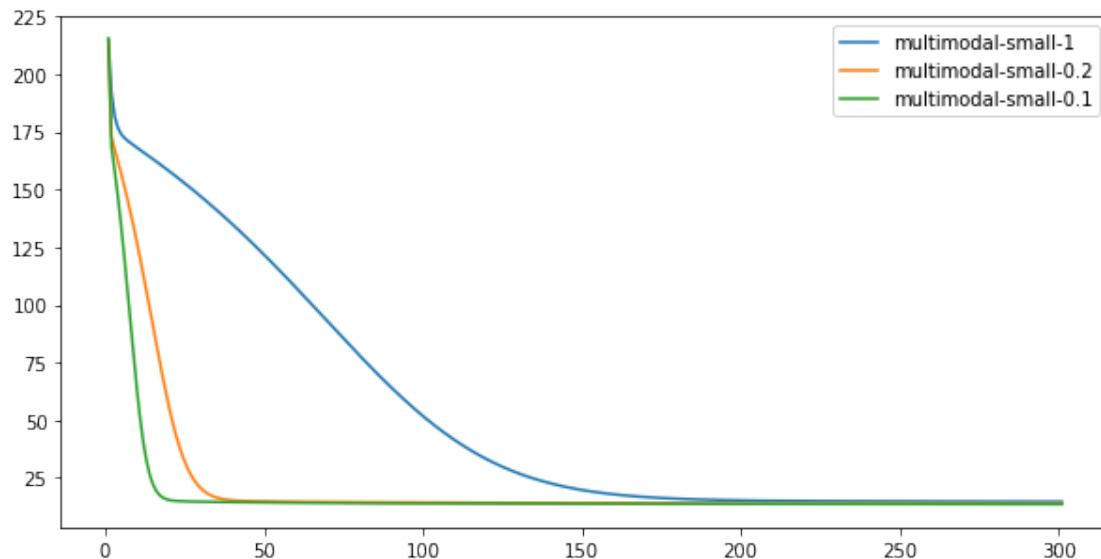
[1090]: <matplotlib.legend.Legend at 0x7f9eb66e05f8>



10% batch najlepszy

## 6.11 Wykres multimodal-small

[1091]: <matplotlib.legend.Legend at 0x7f9eb6724320>



10% batch najlepszy

## 7 Podsumowanie

Używanie tzw stochastycznego gradientu znacznie przyspiesza proces uczenia.

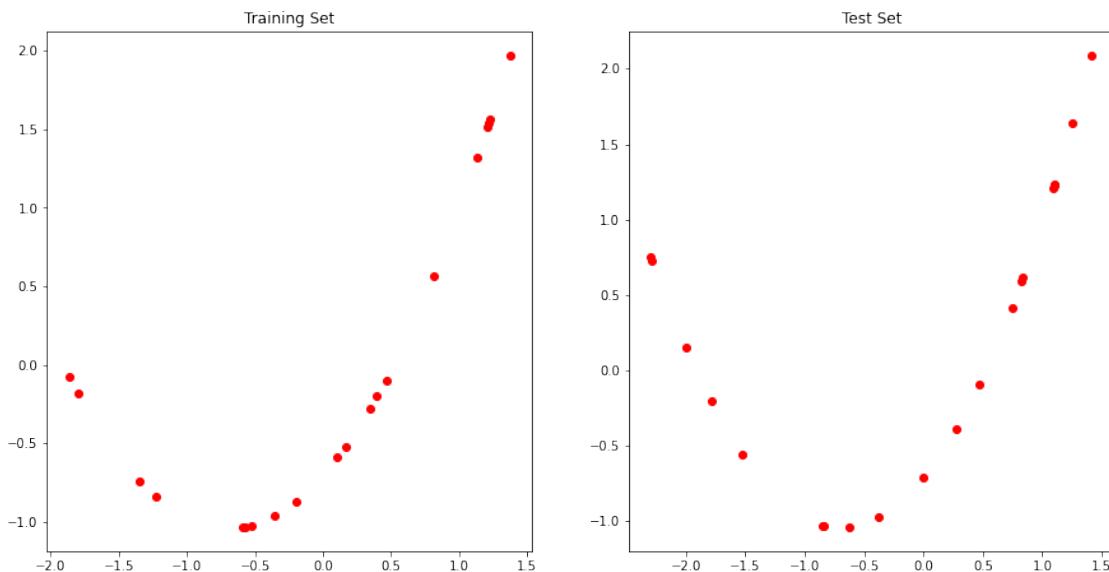
## 8 Eksperyment 3 wpływ momentu na uczenie

Zbadamy wpływ momentu gradientu na proces uczenia. Zastosujemy metodę spadku gradientu z momentem i bez momentu oraz RMSProp. Za batch weźmiemy 10%. Za betę przyjmiemy 0.9 dla momentu i 0.99 dla RMSProp. To będą jedyne zmiany od podstawowych. Będziemy nadal testować na architekturze [8,8]. Zbiory to: square-small, steps-small, multimodal-small

## 9 Hipoteza: poprawią się wyniki

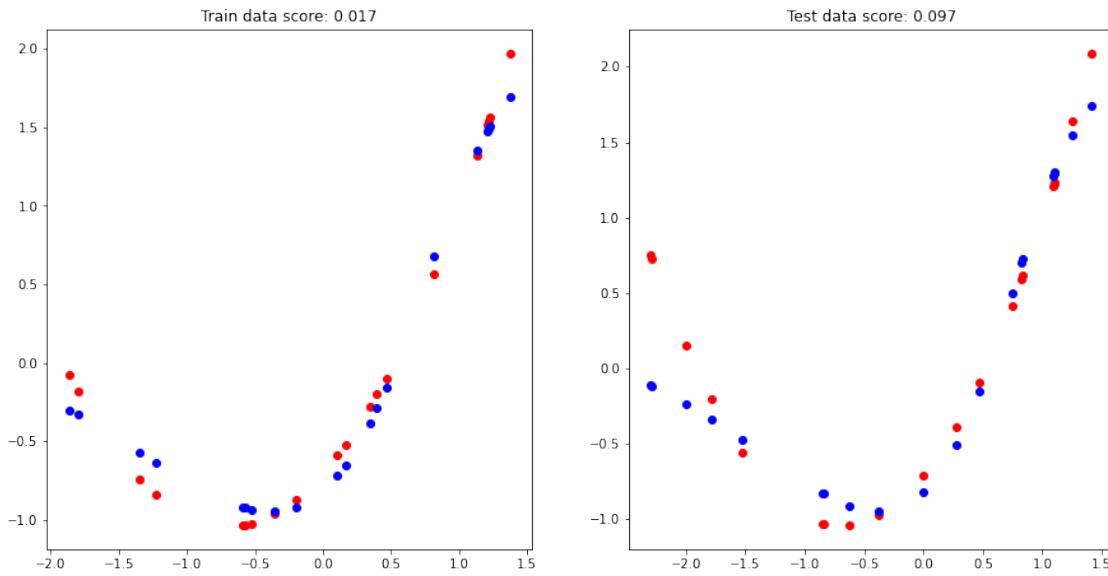
### 9.1 square-small

[1094]: [Text(0.5, 1.0, 'Test Set')]



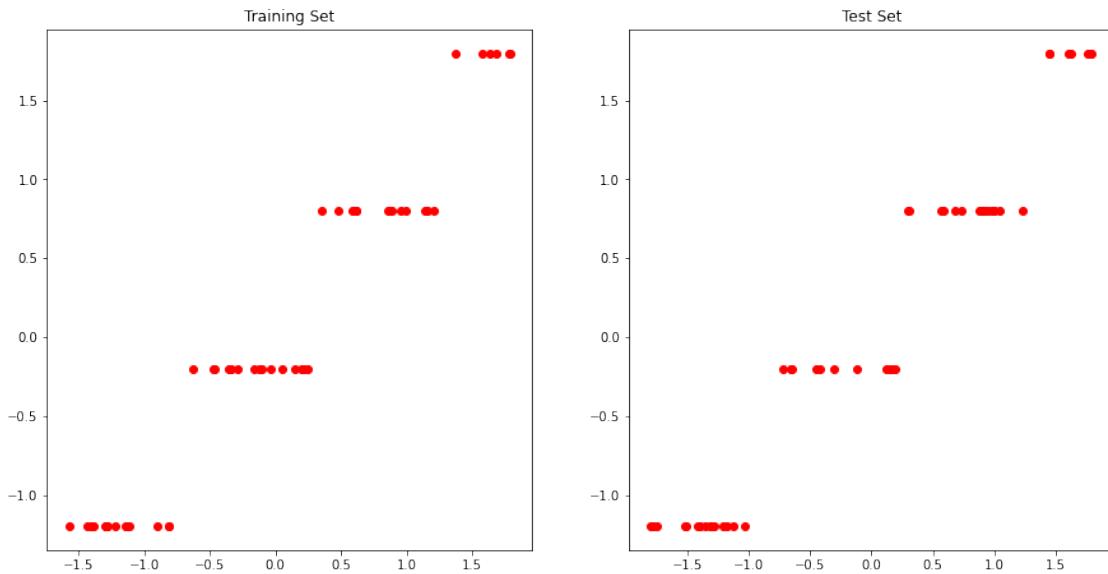
### 9.2 Najlepszy model

[1096]: [Text(0.5, 1.0, 'Test data score: 0.097')]



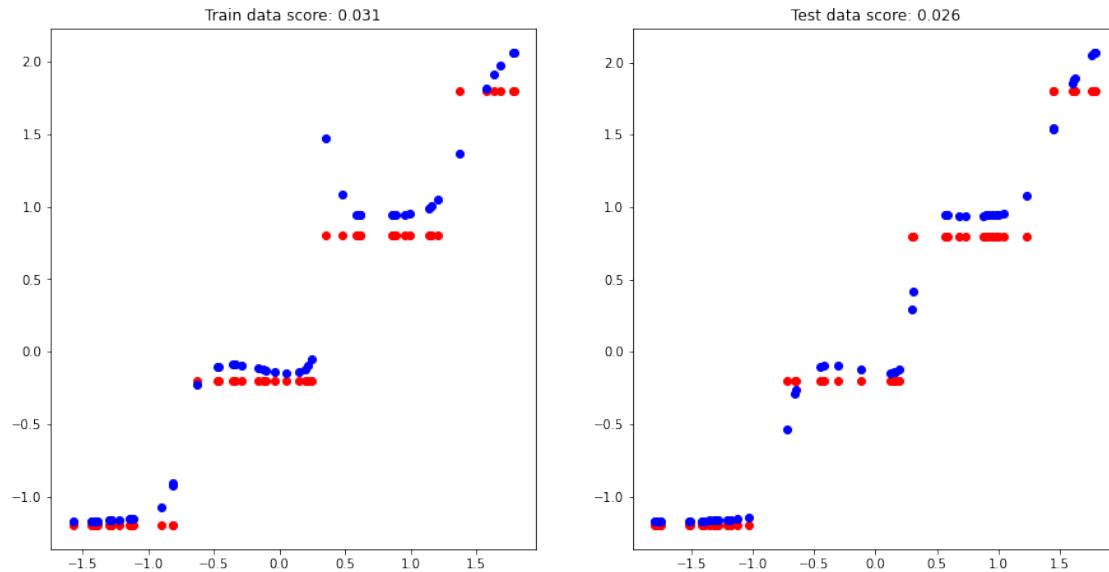
### 9.3 steps-small

[1099]: [Text(0.5, 1.0, 'Test Set')]



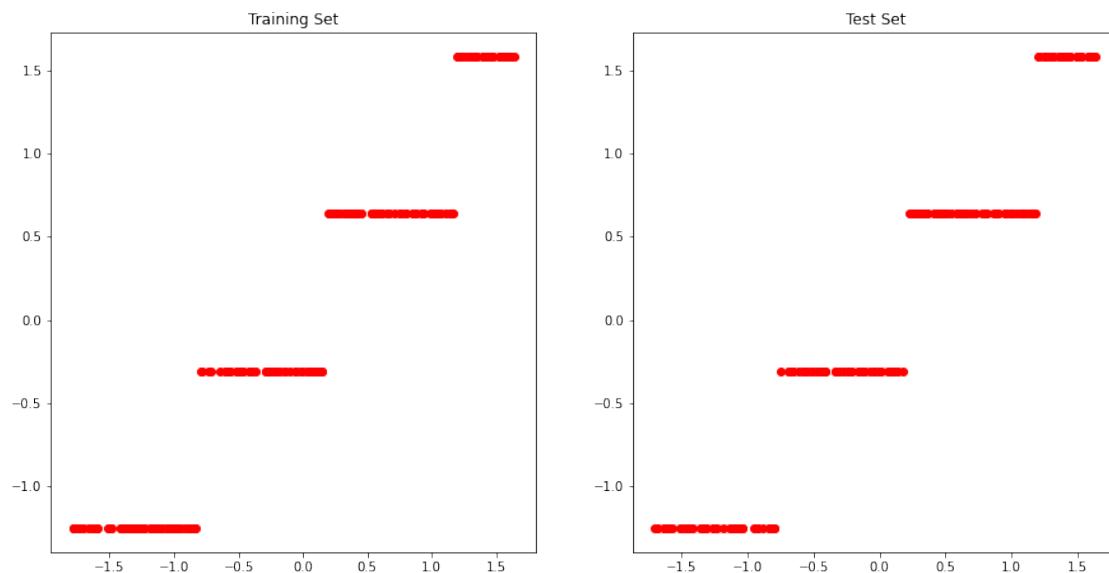
## 9.4 Najlepszy model

[1101]: [Text(0.5, 1.0, 'Test data score: 0.026')]



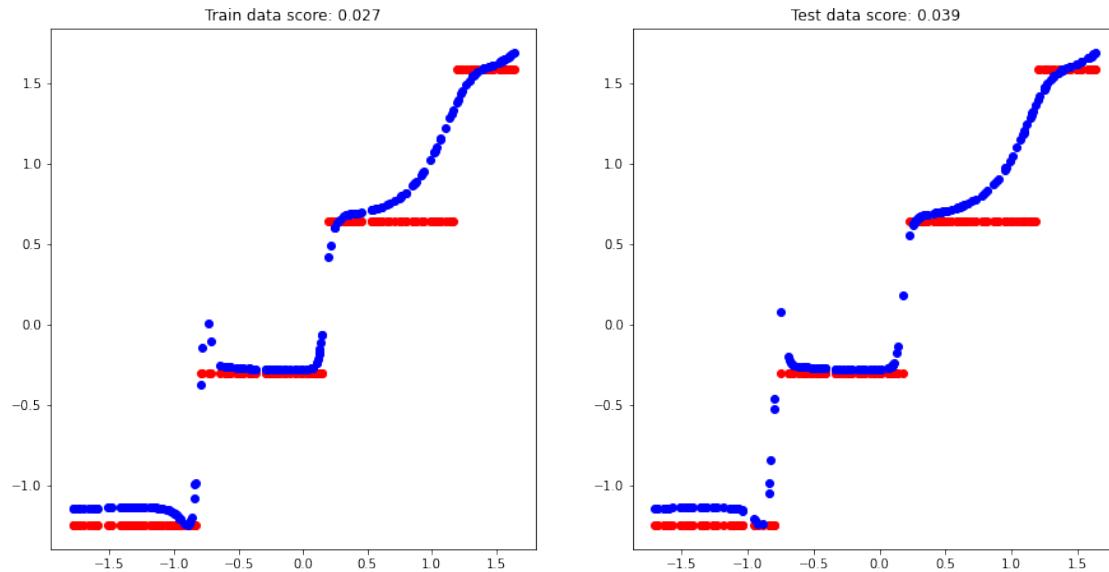
## 9.5 multimodal-small

[1104]: [Text(0.5, 1.0, 'Test Set')]



## 9.6 Najlepszy model

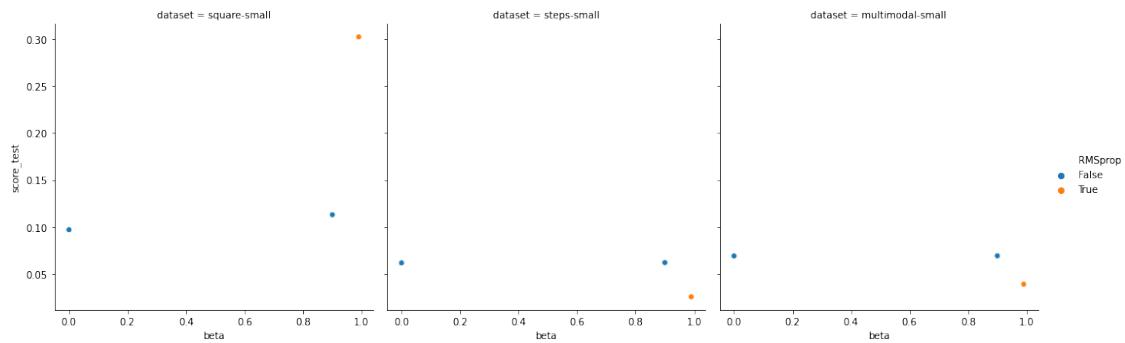
[1106]: [Text(0.5, 1.0, 'Test data score: 0.039')]



## 9.7 Wykres wyniku od momentu

### 9.7.1 MSE

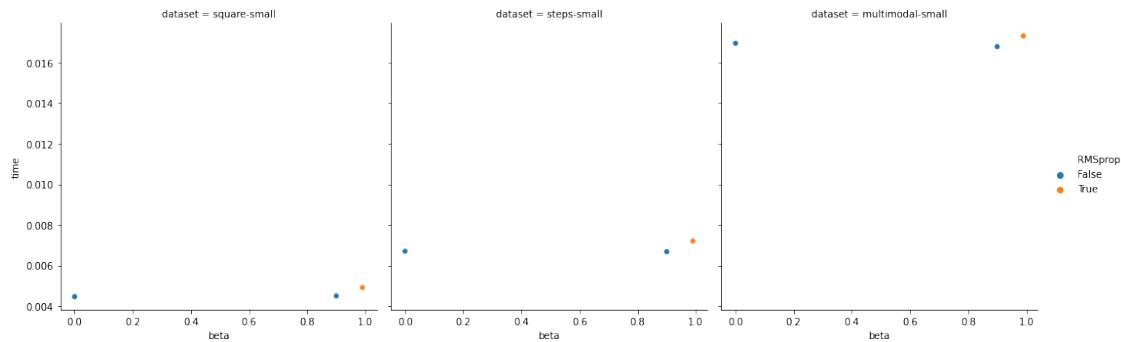
[1110]: <seaborn.axisgrid.FacetGrid at 0x7f9eb83d6128>



RMSprop daje albo relewacyjne wyniki, albo bardzo słabe, między gradientem z momentem i bez momentu nie widać takiej różnicy.

## 9.8 Wykres czasu od momentu

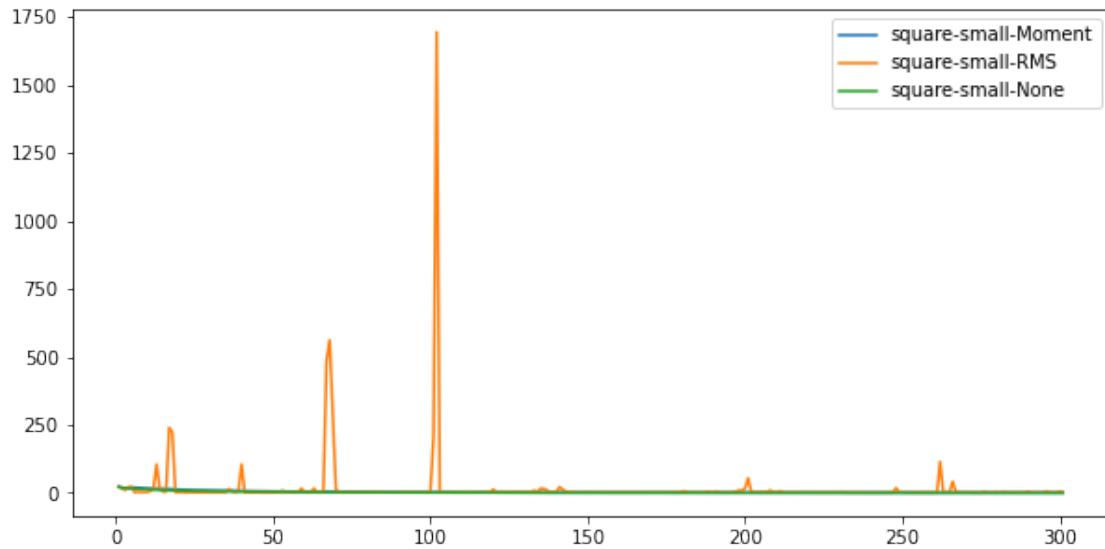
[1111]: <seaborn.axisgrid.FacetGrid at 0x7f9eb6039e80>



RMSprop wymaga więcej czasu, ale proporcjonalnie mniej dla większych danych.

## 9.9 Wykres błędów square-small

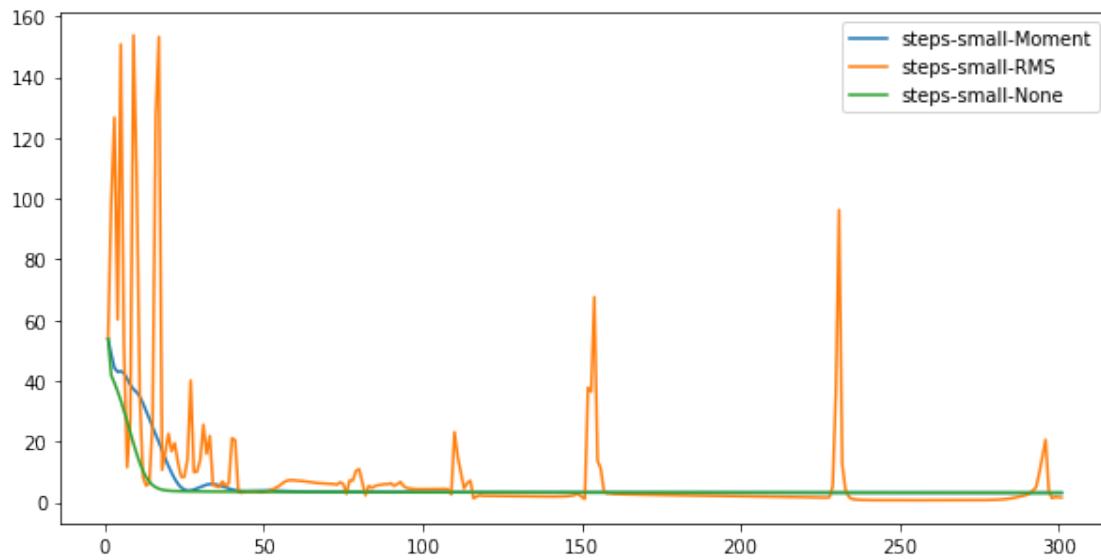
[1216]: <matplotlib.legend.Legend at 0x7f9eb2be67f0>



RMSprop jest niestabilny, moment uczy się wolniej niż brak.

## 9.10 Wykres błędów steps-small

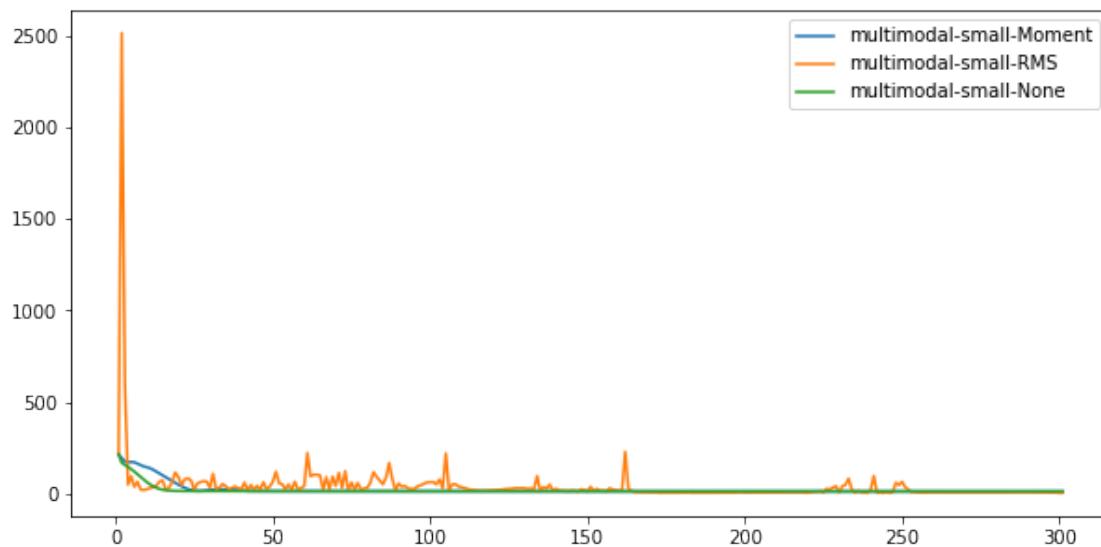
[1113]: <matplotlib.legend.Legend at 0x7f9eb5ef2e10>



RMSprop jest niestabilny, moment uczy się wolniej niż brak.

## 9.11 Wykres multimodal-small

[1114]: <matplotlib.legend.Legend at 0x7f9eb5ef86a0>



RMSprop jest niestabilny, moment uczy się wolniej niż brak.

## 10 Podsumowanie

RMSprop może dawać dobre rezultaty, ale ma chaotyczne zachowanie.

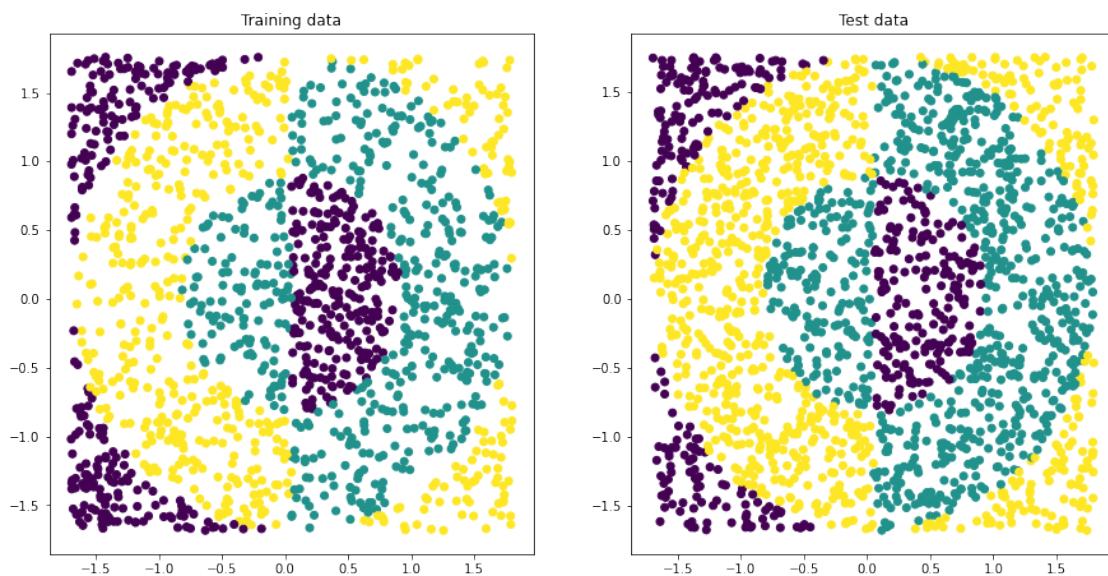
## 11 Ekperyment 4 wpływ softmax w procesie uczenia.

Sprawdzimy czy przy rozwiązywaniu zadań klasyfikacji używanie funkcji softmax na ostatniej warstwie i kross entropii jest lepsze niż MSE. Będziemy testować na zbiorach: rings3-regular, easy i xor3. Testy przeprowadzimy na architekturze [8,8] używając 10% batch i momentu 0.9.

## 12 Hipoteza: softmax okaże się lepszy.

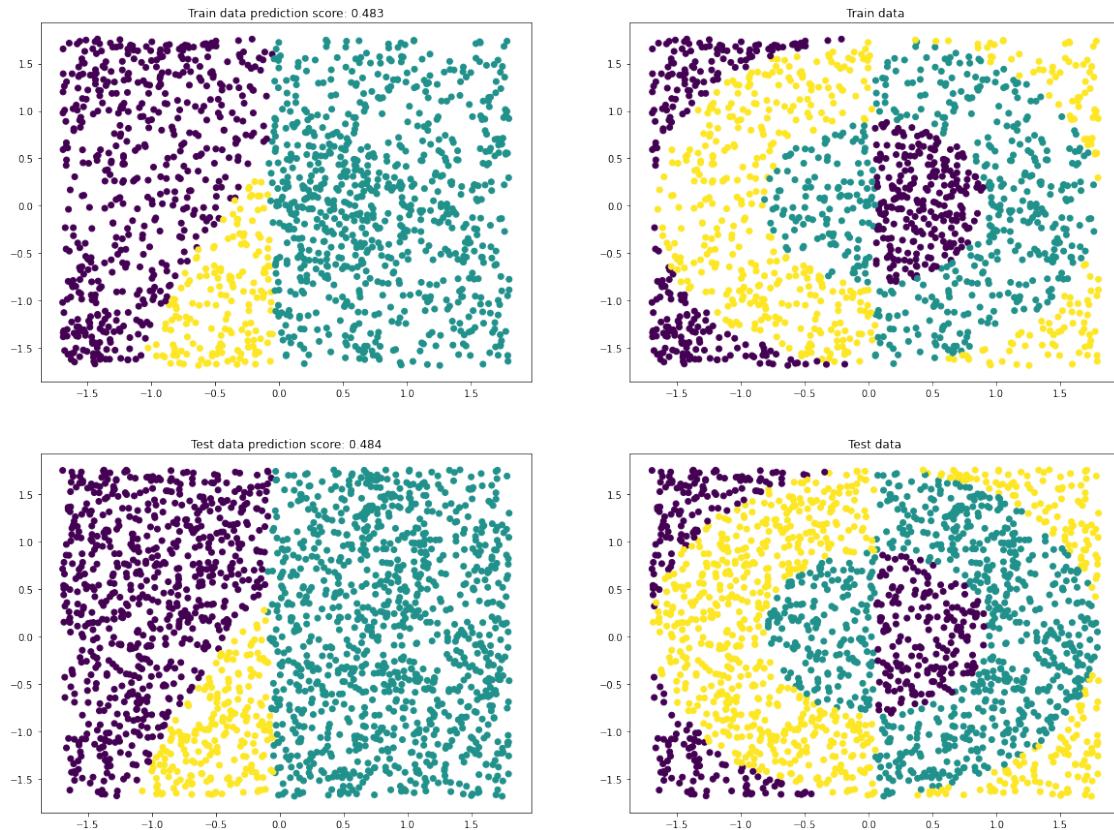
### 12.1 rings5-regular

```
[1117]: [Text(0.5, 1.0, 'Test data')]
```



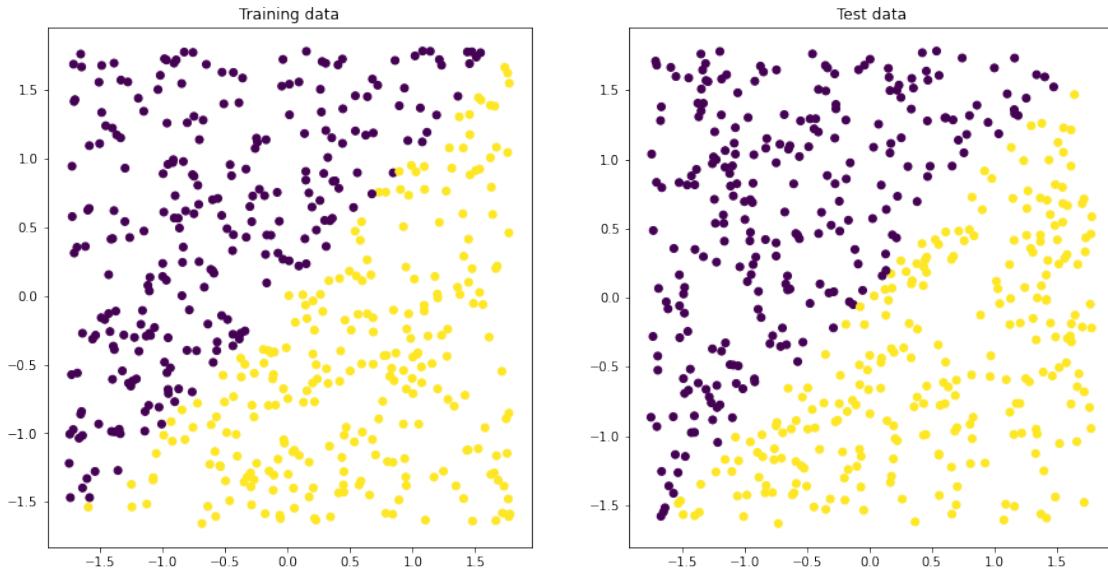
## 12.2 Najlepszy model

[1119]: [Text(0.5, 1.0, 'Test data')]



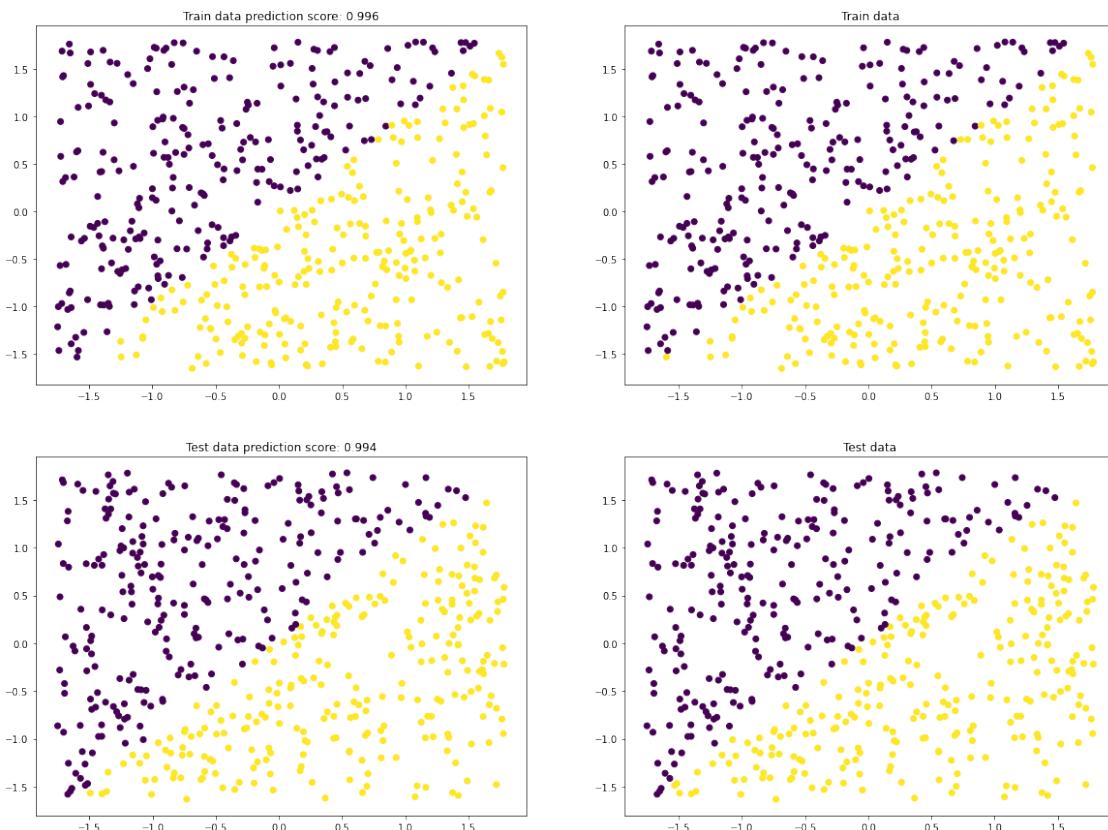
## 12.3 easy

[1122]: [Text(0.5, 1.0, 'Test data')]



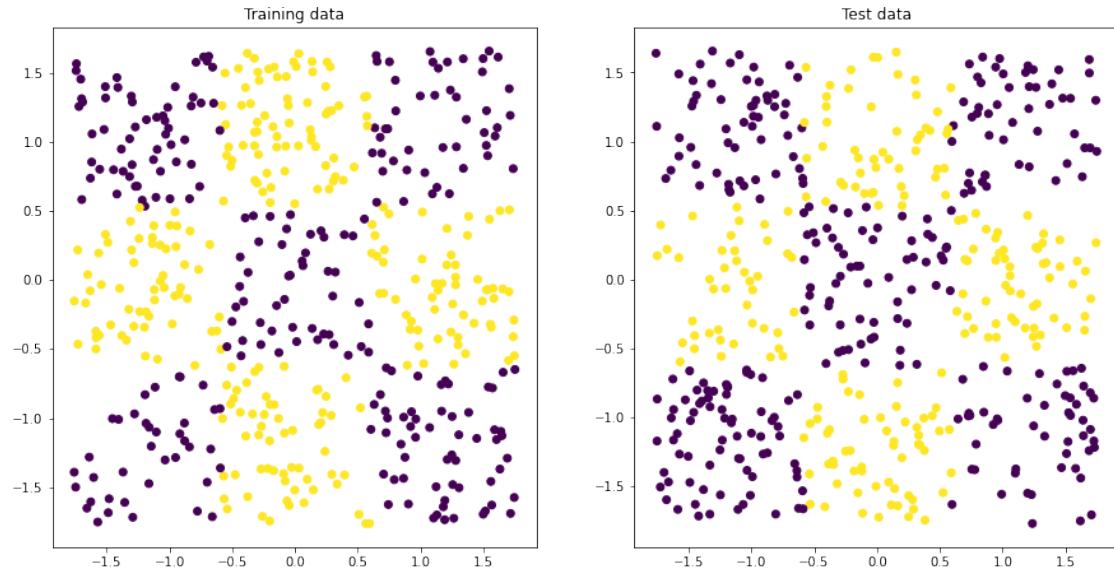
## 12.4 Najlepszy model

[1124]: [Text(0.5, 1.0, 'Test data')]



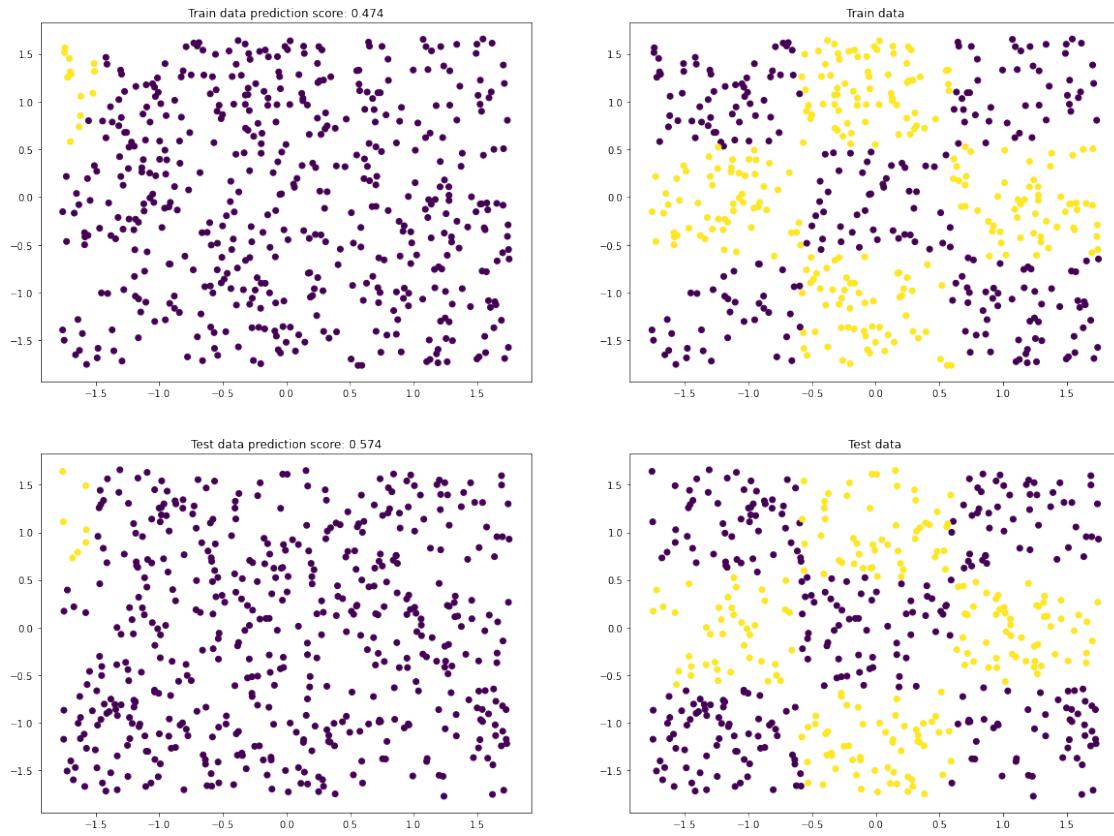
## 12.5 xor3

```
[1127]: [Text(0.5, 1.0, 'Test data')]
```



## 12.6 Najlepszy model

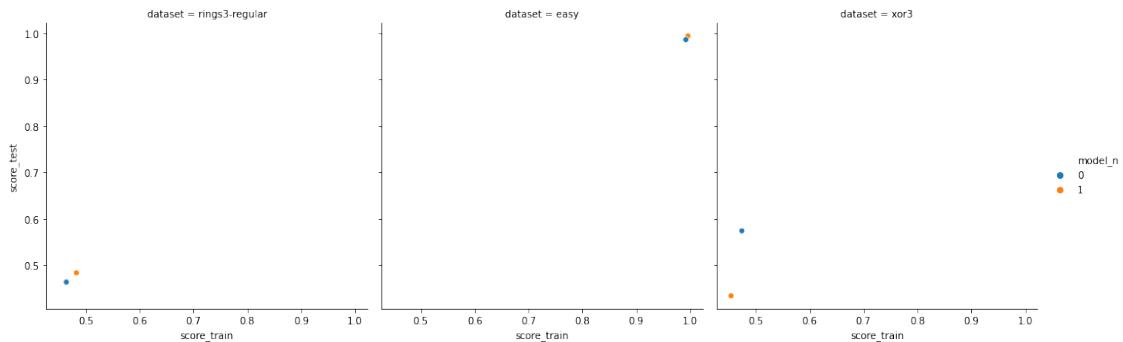
```
[1129]: [Text(0.5, 1.0, 'Test data')]
```



## 12.7 Wykres wyniku od rodzaju funkcji (softmax na pomarańczowo)

### 12.8 Accuracy

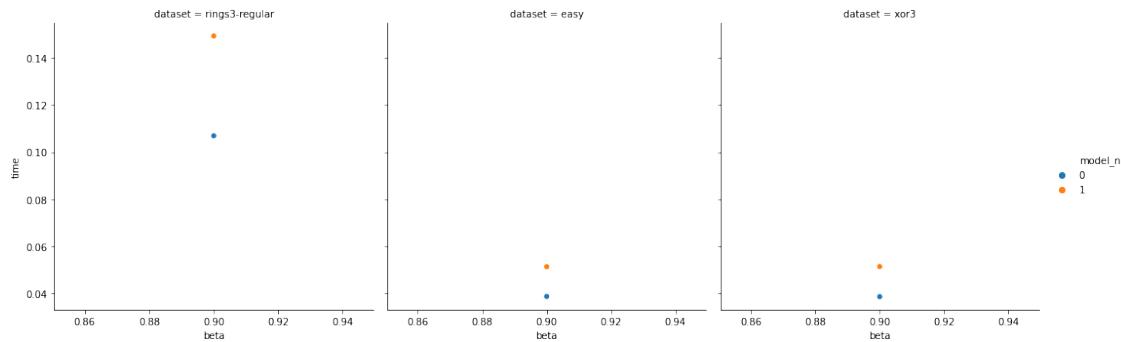
[1132]: <seaborn.axisgrid.FacetGrid at 0x7f9eb58591d0>



softmax daje ogólnie lepsze wyniki, ale z xor miał problemy.

## 12.9 Wykres czasu

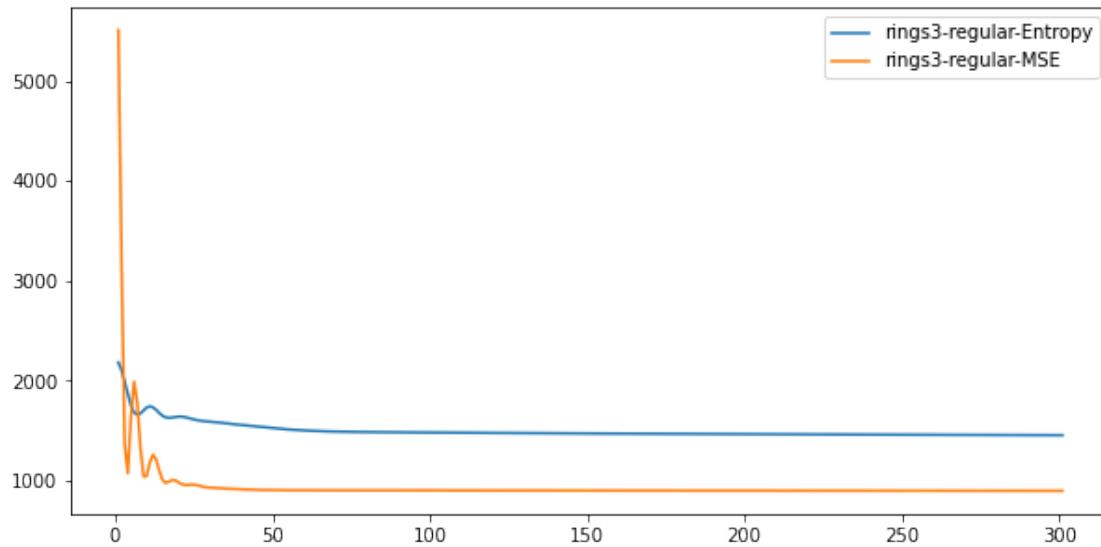
[1133]: <seaborn.axisgrid.FacetGrid at 0x7f9eb56d3320>



softmax wymaga więcej czasu, ale wraz ze wzrostem zbioru danych proporcjonalnie mniej.

## 12.10 Wykres błędów ring3-regular

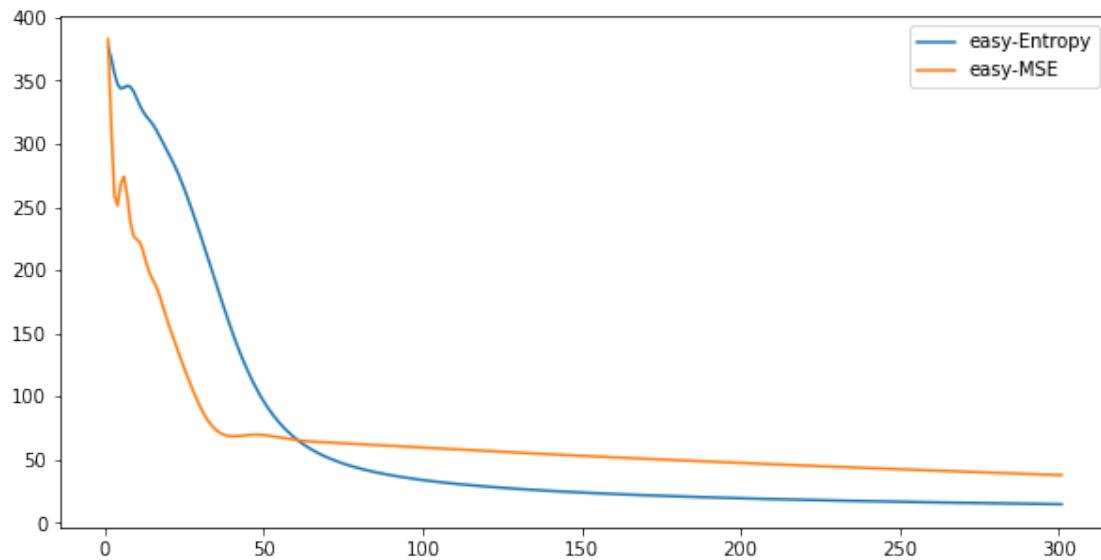
[1134]: <matplotlib.legend.Legend at 0x7f9eb5491c88>



Entropia ma bardziej gładki kształt.

## 12.11 Wykres błędów easy

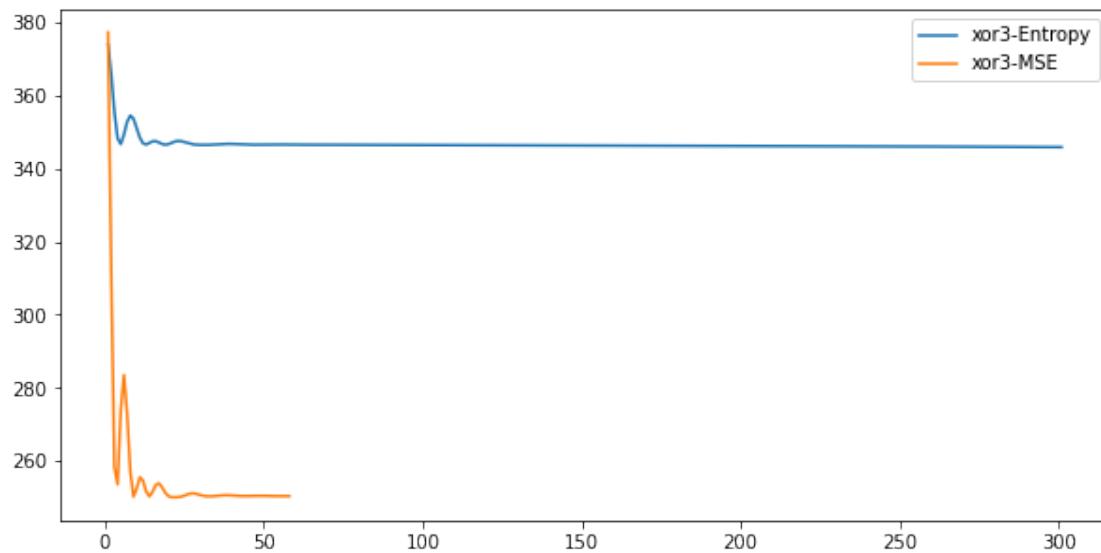
[1135]: <matplotlib.legend.Legend at 0x7f9eb5406e10>



Entropia ma bardziej gładki kształt.

## 12.12 Wykres xor3

[1136]: <matplotlib.legend.Legend at 0x7f9eb5515470>



MSE wcześniej zakończyło naukę

## 13 Podsumowanie

W zadaniach klasyfikacji softmax jest lepszy.

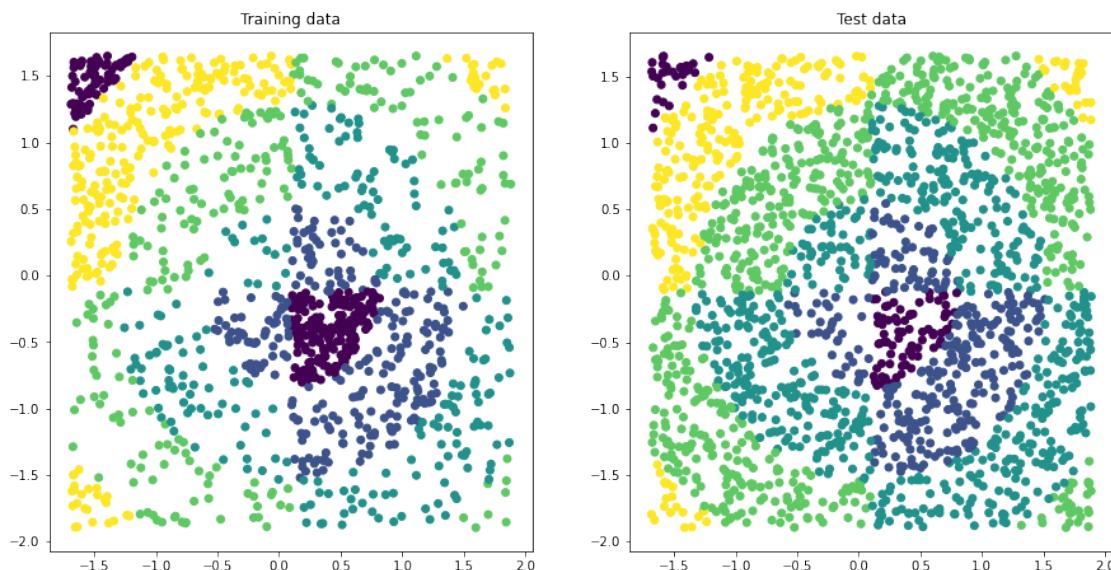
## 14 Eksperyment 5 Różne funkcje aktywacji

Będziemy testować różne funkcje aktywacji: sigmoid, tanh, ReLU i liniową w zależności od liczby warstw i neuronów. Będziemy testować na architekturach [5],[8],[5,5],[8,8],[5,5,5],[8,8,8]. Batch 10%, zaś beta=0.9.

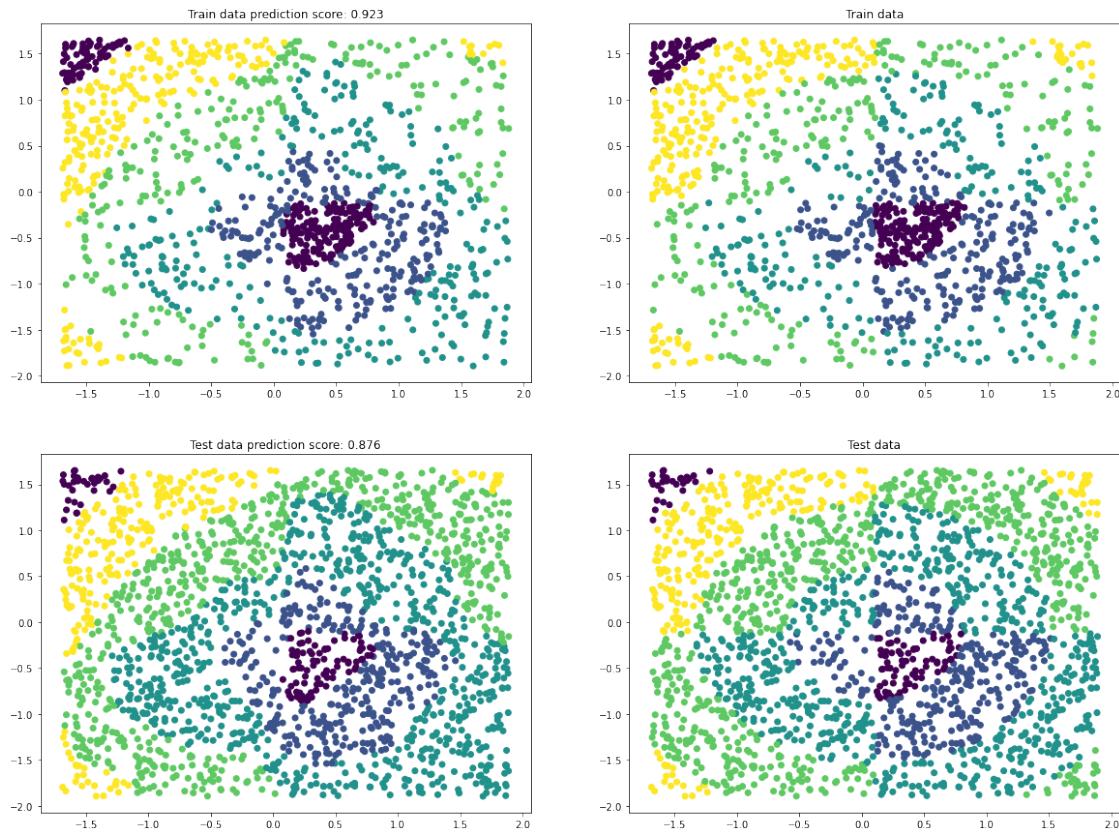
## 15 Hipoteza: sigmoid nie będzie najlepszy

## 16 rings5-regular

[1139]: [Text(0.5, 1.0, 'Test data')]

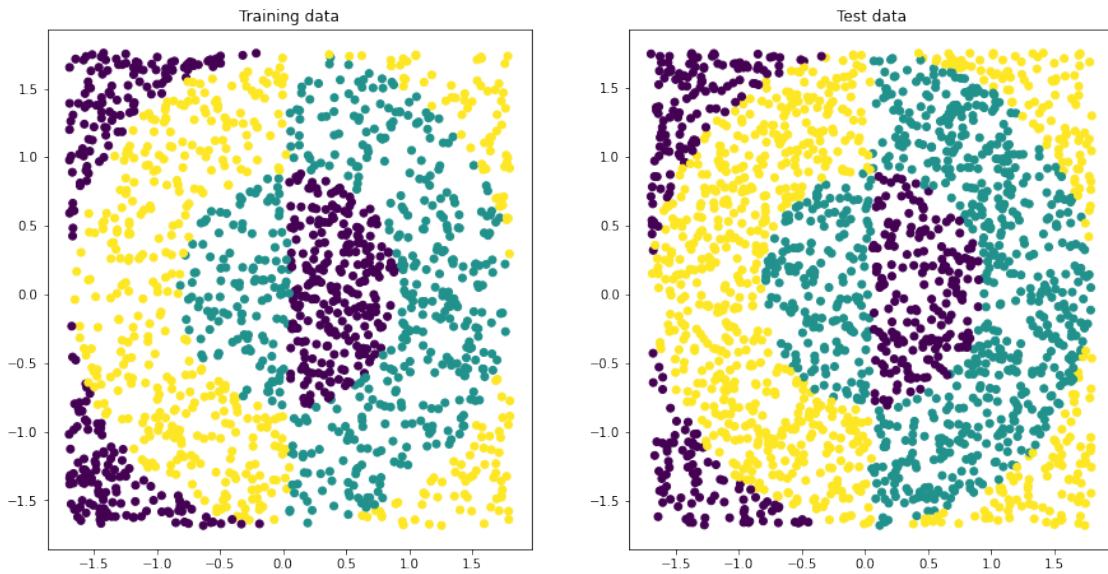


[1141]: [Text(0.5, 1.0, 'Test data')]



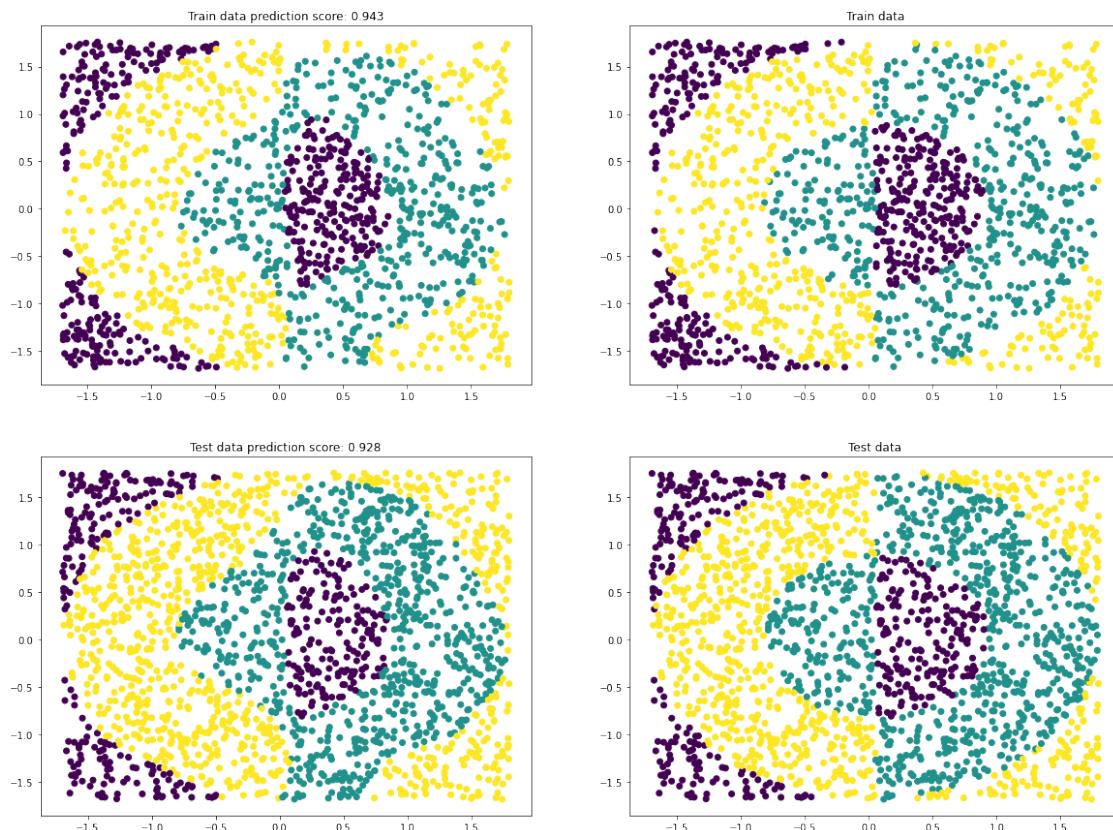
## 17 rings3-regular

```
[1144]: [Text(0.5, 1.0, 'Test data')]
```



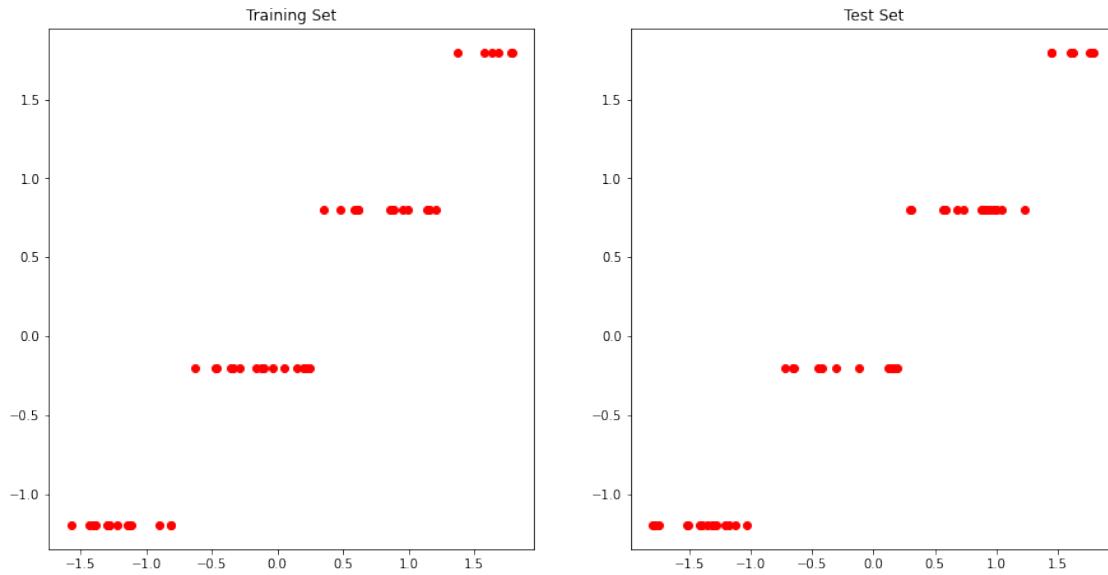
## 17.1 Najlepszy model

[1146]: [Text(0.5, 1.0, 'Test data')]



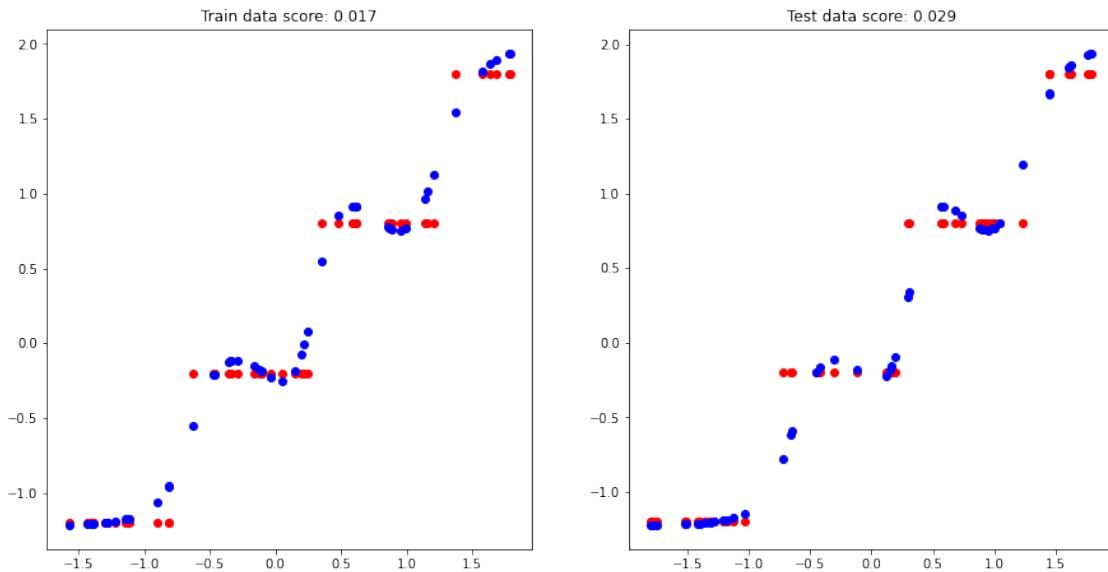
## 18 steps-small

[1149]: [Text(0.5, 1.0, 'Test Set')]



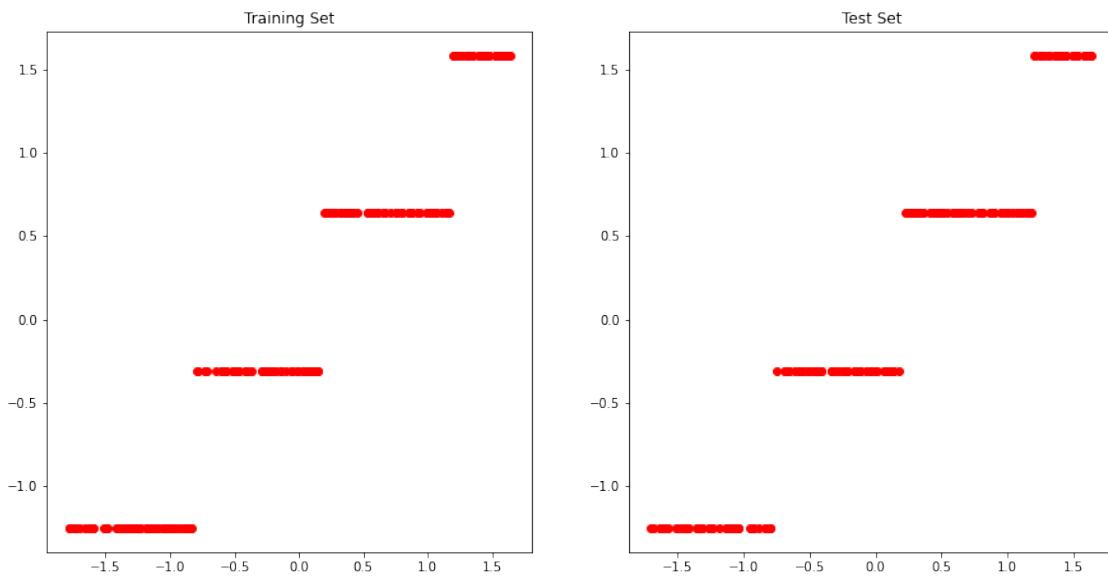
### 18.1 Najlepszy model

[1151]: [Text(0.5, 1.0, 'Test data score: 0.029')]



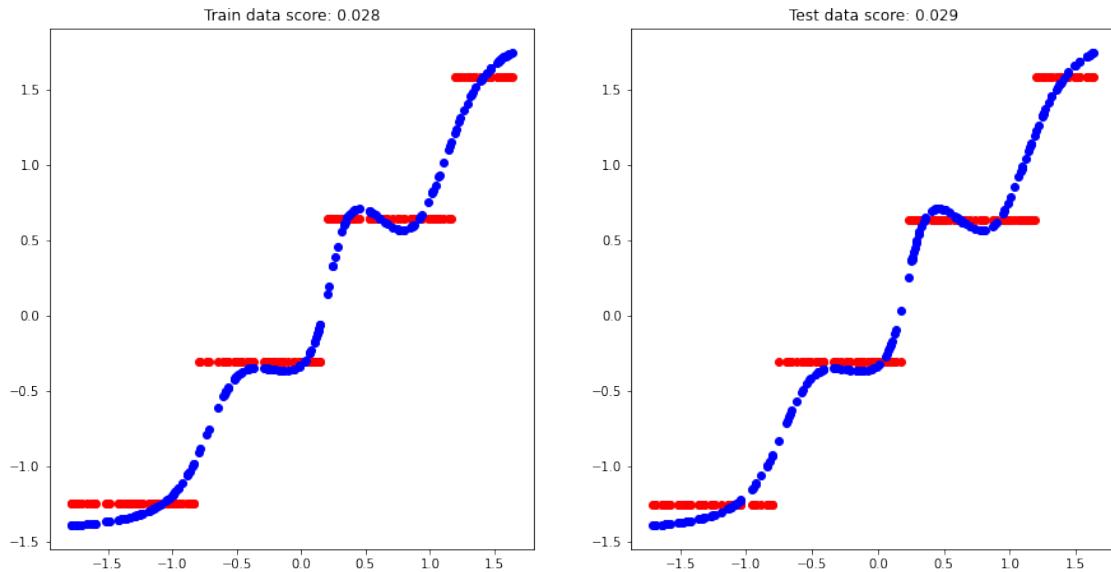
## 19 multimodal-small

[1154]: [Text(0.5, 1.0, 'Test Set')]



## 19.1 Najlepszy model

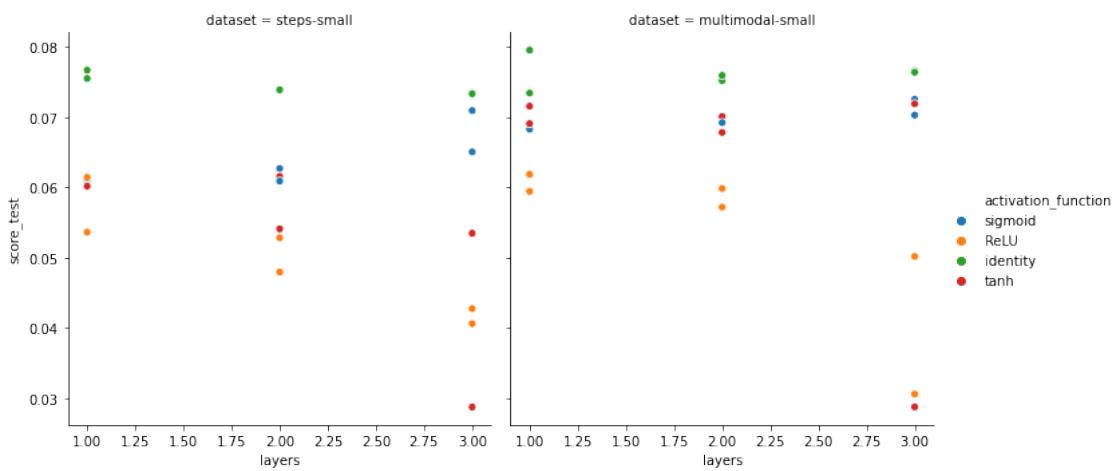
[1156]: [Text(0.5, 1.0, 'Test data score: 0.029')]



## 19.2 Wyniki od liczby warstw i funkcji aktywacji

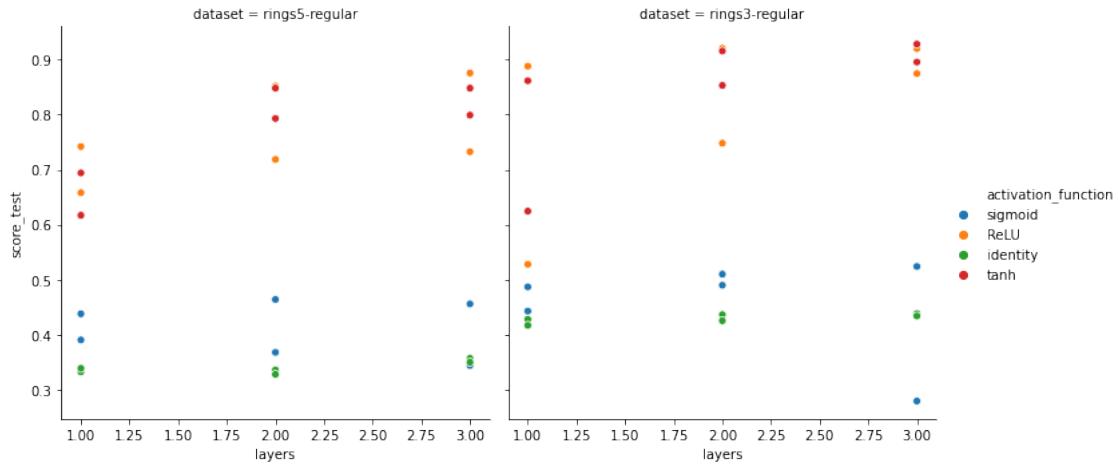
### 19.3 MSE

[1161]: <seaborn.axisgrid.FacetGrid at 0x7f9eb4be1710>



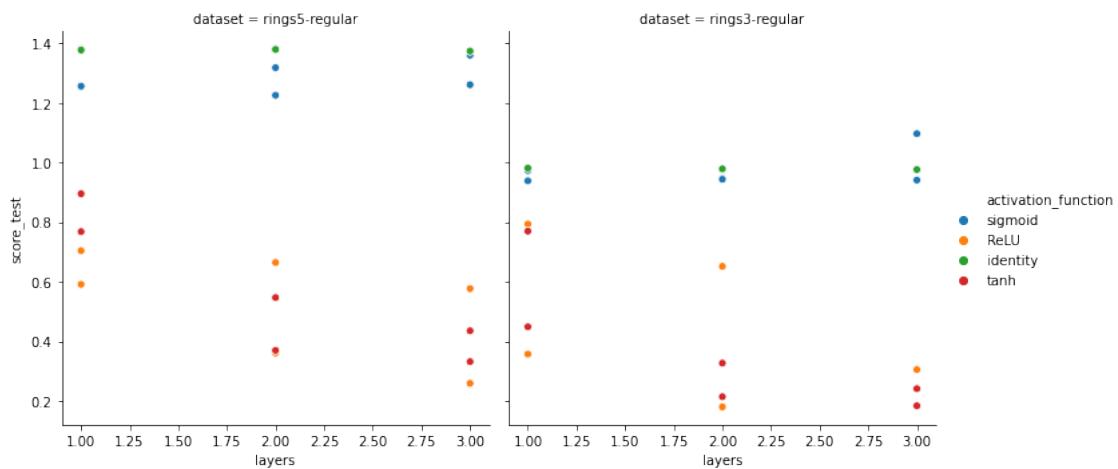
## 19.4 Accuracy

[1162]: <seaborn.axisgrid.FacetGrid at 0x7f9eb4d86ef0>



## 19.5 Entropy

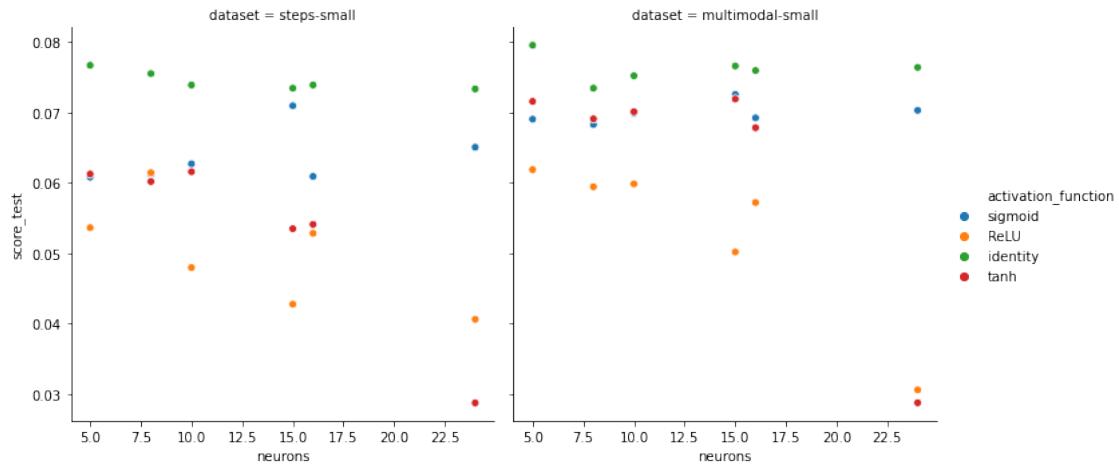
[1163]: <seaborn.axisgrid.FacetGrid at 0x7f9eb4a2aa90>



## 19.6 Wyniki od liczby neuronów i funkcji aktywacji

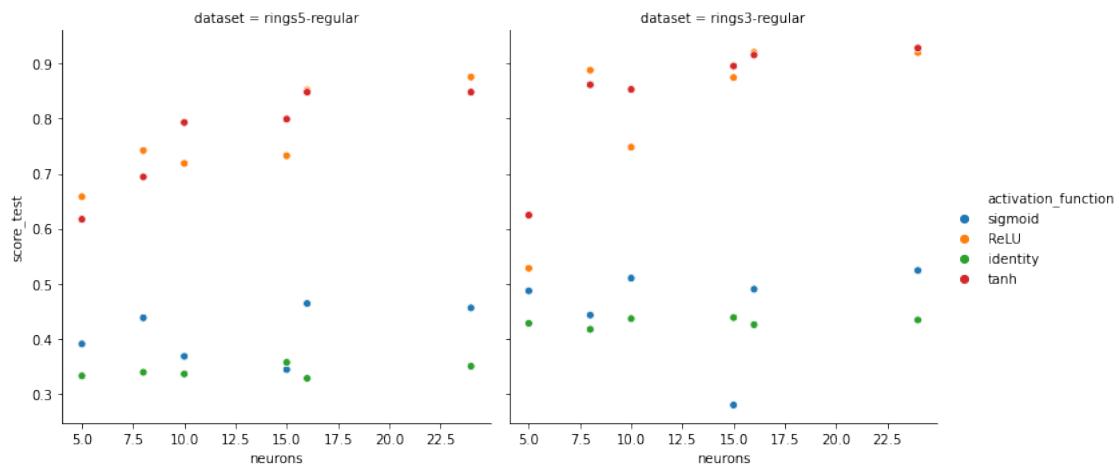
### 19.7 MSE

[1164]: <seaborn.axisgrid.FacetGrid at 0x7f9eb4936240>



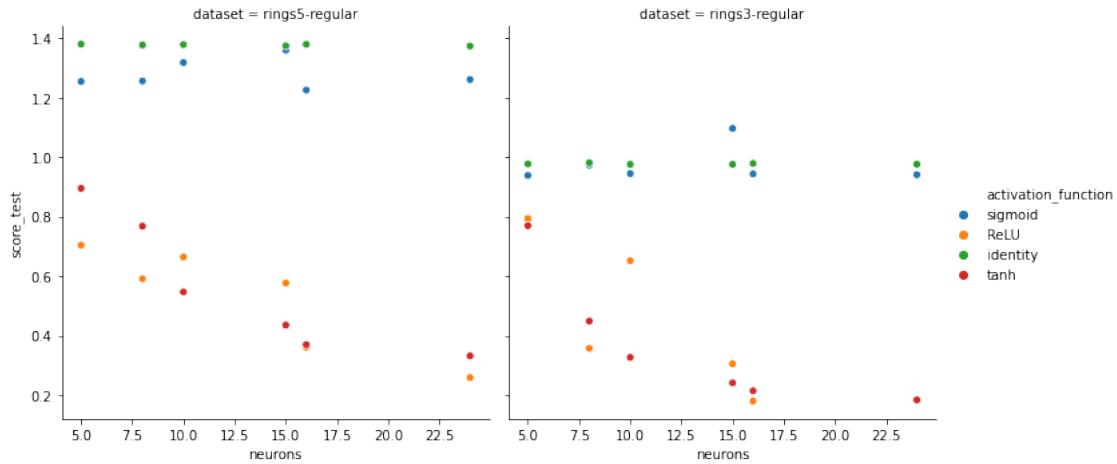
### 19.8 Accuracy

[1165]: <seaborn.axisgrid.FacetGrid at 0x7f9eb494db38>



## 19.9 Entropy

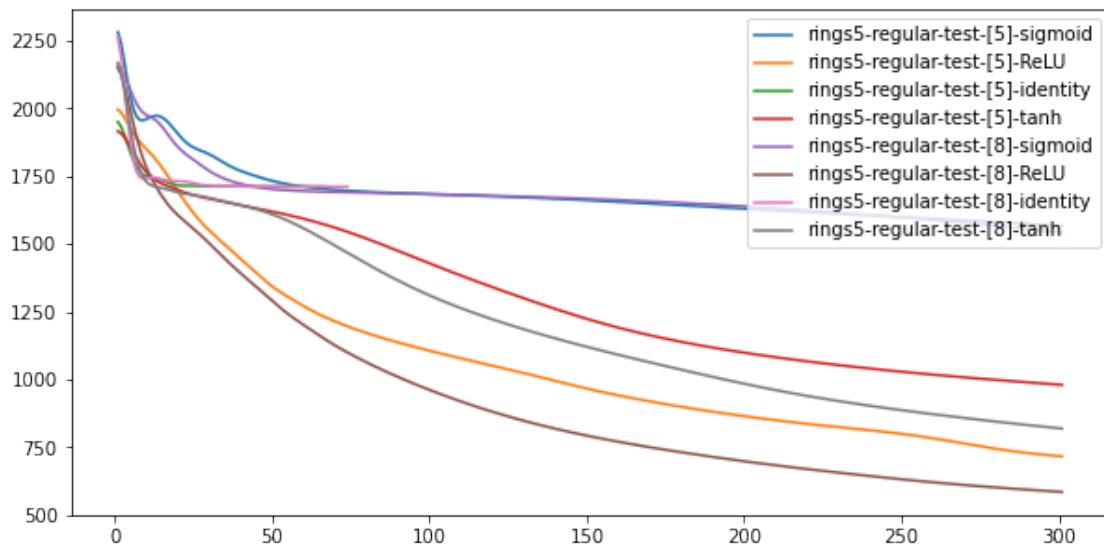
[1166]: <seaborn.axisgrid.FacetGrid at 0x7f9eb4643be0>



Ogólnie to tanh i ReLU dają podobne wyniki i są lepsze niż liniowa i sigmoid.

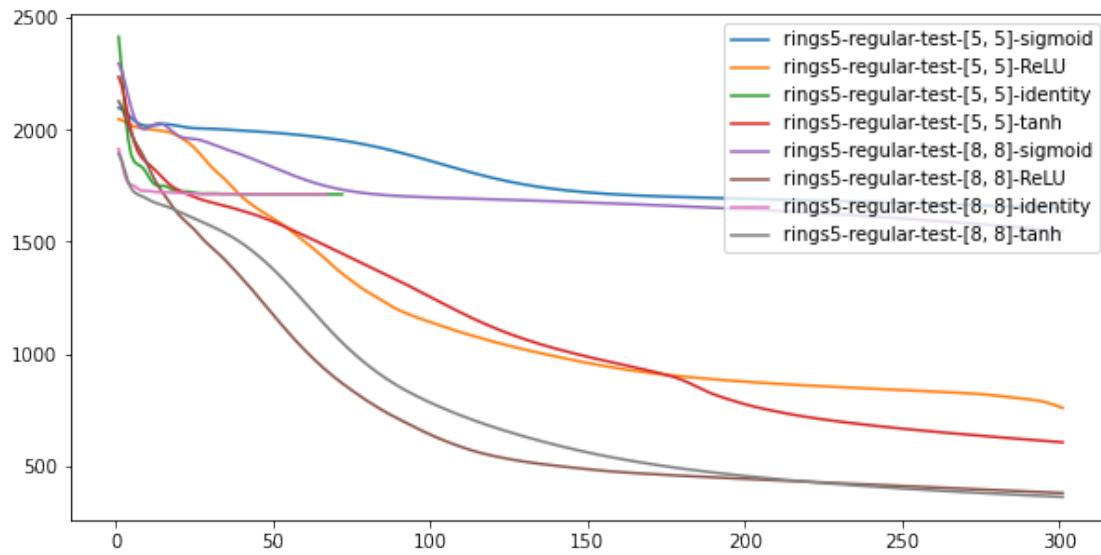
## 19.10 Wykres błędów rings5-regular 1 warstwa

[1167]: <matplotlib.legend.Legend at 0x7f9eb459d390>



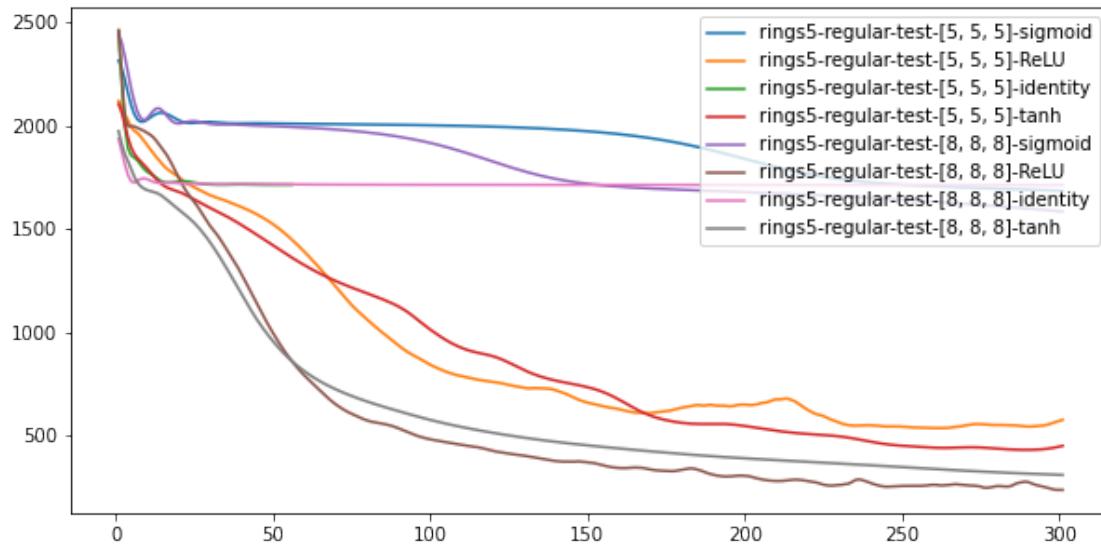
## 19.11 Wykres błędów rings5-regular 2 warstwy

[1168]: <matplotlib.legend.Legend at 0x7f9eb44c4278>



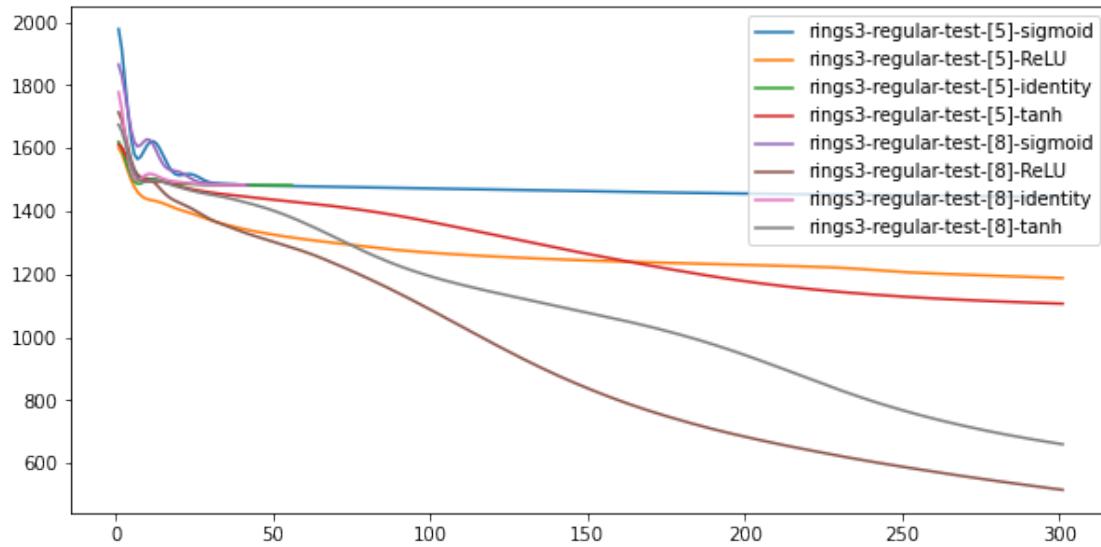
## 19.12 Wykres błędów rings5-regular 3 warstwy

[1169]: <matplotlib.legend.Legend at 0x7f9eb44602b0>



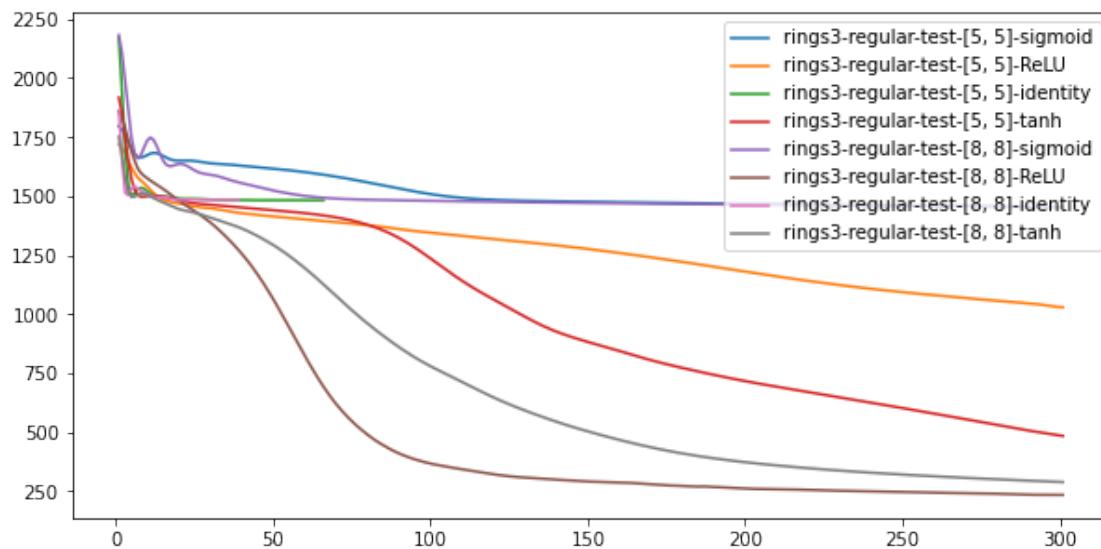
### 19.13 Wykres błędów rings3-regular 1 warstwa

[1170]: <matplotlib.legend.Legend at 0x7f9eb452ea58>



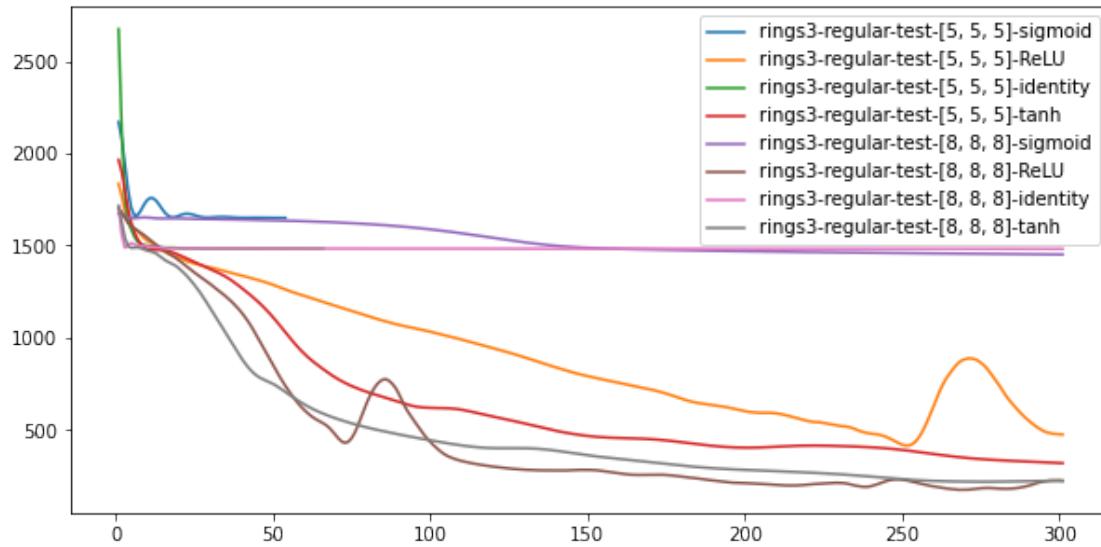
### 19.14 Wykres błędów rings3-regular 2 warstwy

[1171]: <matplotlib.legend.Legend at 0x7f9eb452ebe0>



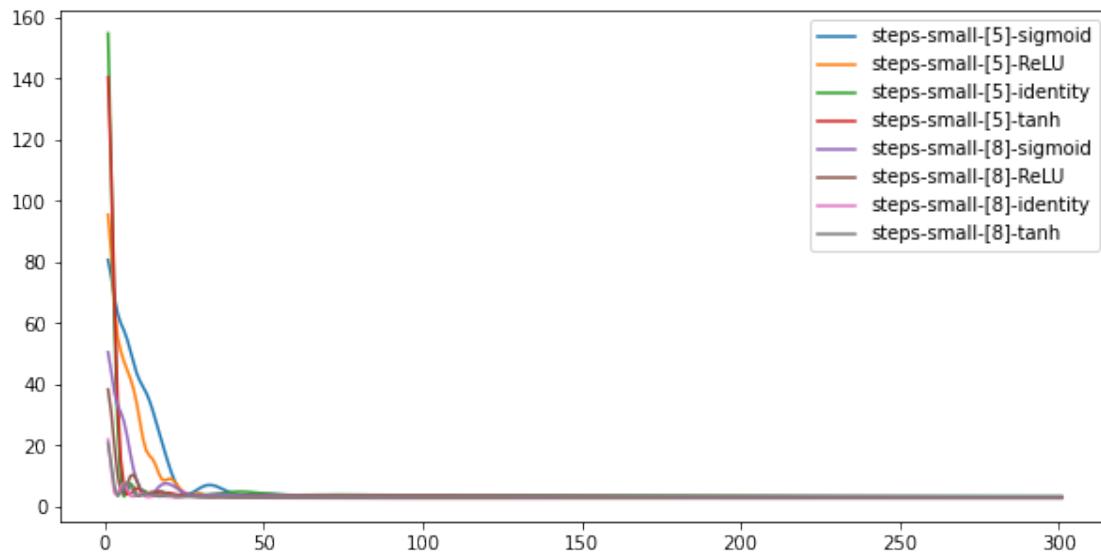
## 19.15 Wykres błędów rings3-regular 3 warstwy

[1172]: <matplotlib.legend.Legend at 0x7f9eb42a7ef0>



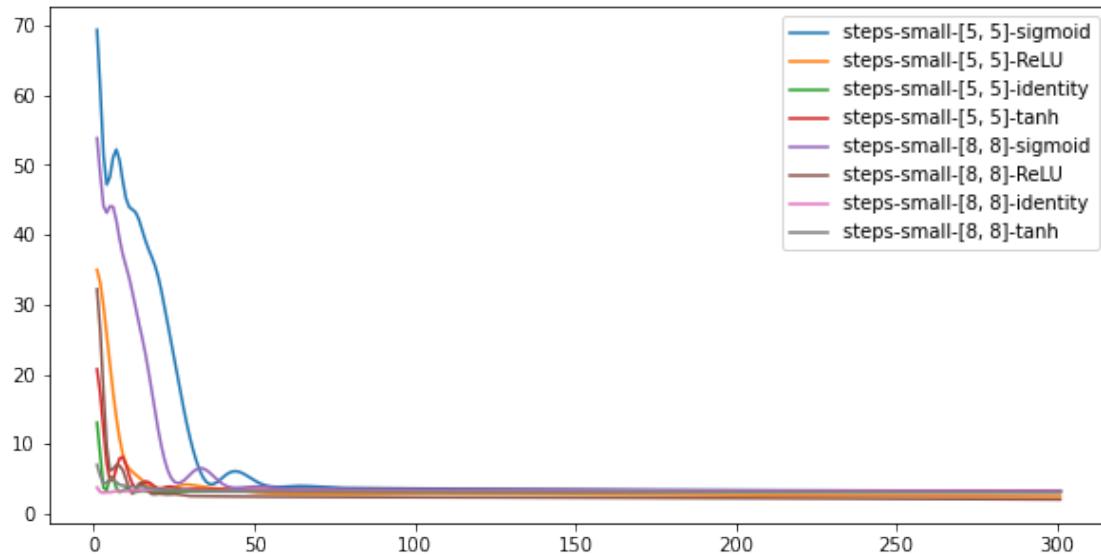
## 19.16 Wykres błędów steps-small 1 warstwa

[1173]: <matplotlib.legend.Legend at 0x7f9eb421c4e0>



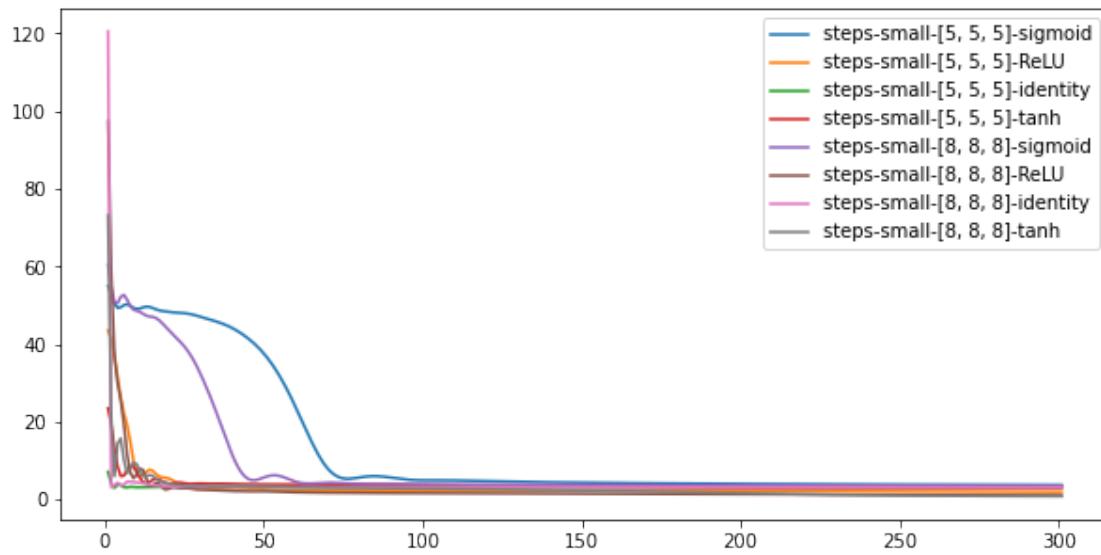
## 19.17 Wykres błędów steps-small 2 warstwy

[1174]: <matplotlib.legend.Legend at 0x7f9eb428f1d0>



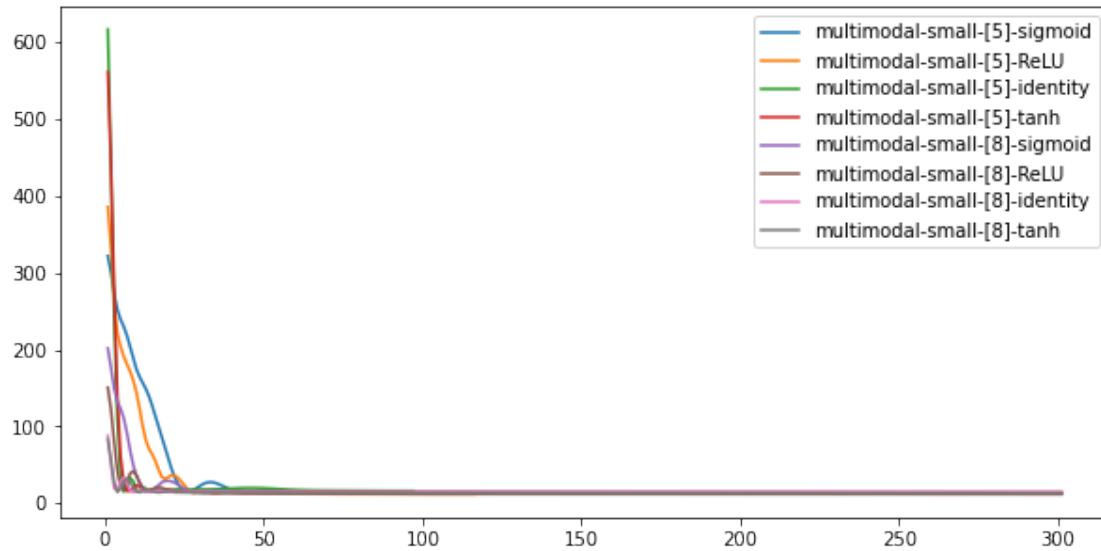
## 19.18 Wykres błędów steps-small 3 warstwy

[1175]: <matplotlib.legend.Legend at 0x7f9eb4088d68>



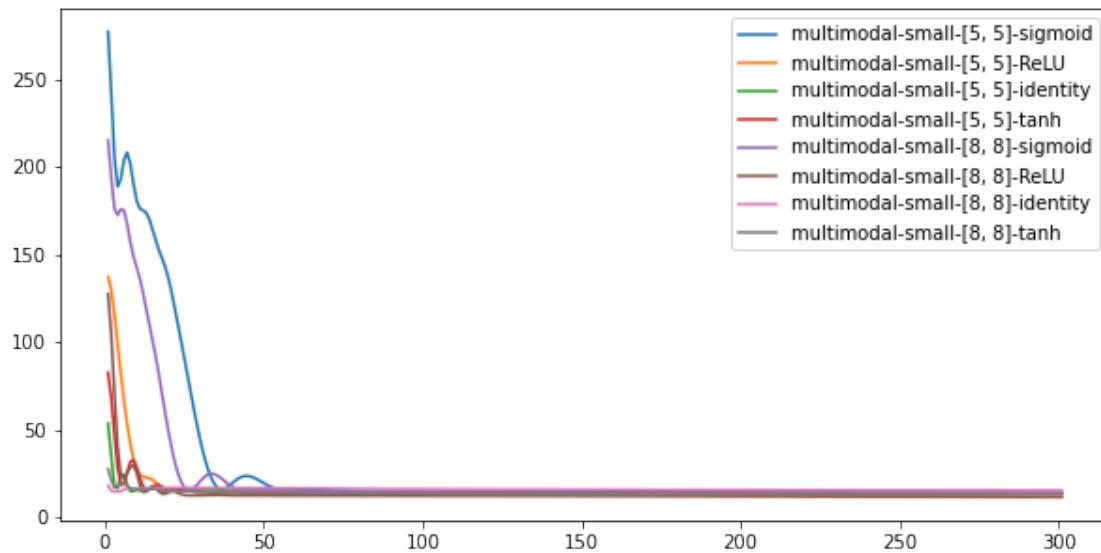
## 19.19 Wykres błędów multimodal-small 1 warstwa

[1176]: <matplotlib.legend.Legend at 0x7f9eb40260f0>



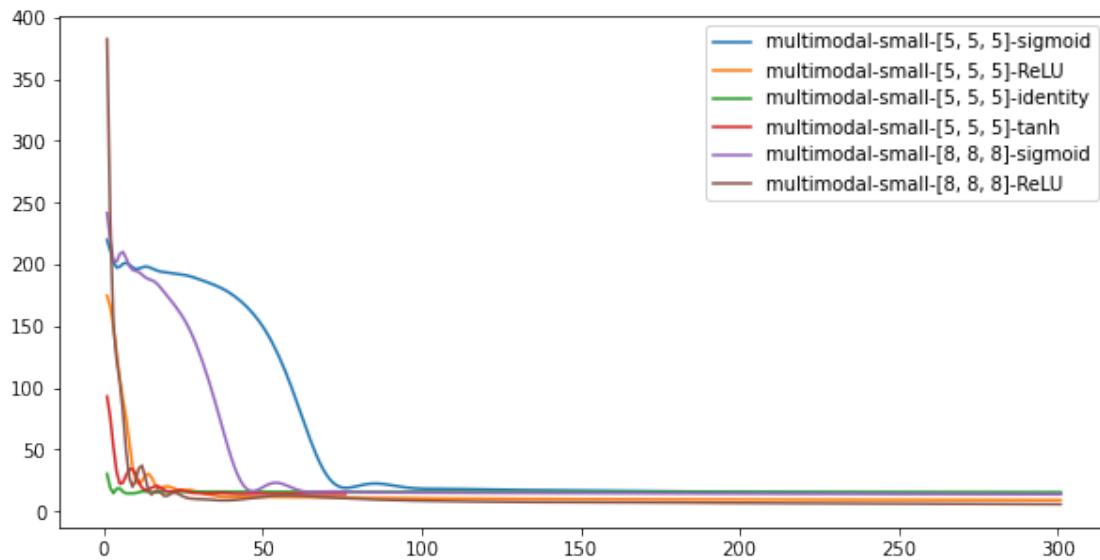
## 19.20 Wykres błędów multimodal-small 2 warstwy

[1177]: <matplotlib.legend.Legend at 0x7f9eb407dac8>



## 19.21 Wykres błędów multimodal-small 3 warstwy

[1178]: <matplotlib.legend.Legend at 0x7f9eb3edf390>



## 20 Podsumowanie

Widzimy że ReLU może zachowywać się niestabilnie, to znaczy że może mieć okresowo wzrosty błędu, zaś tanh jest stabilniejszy i daje podobne rezultaty

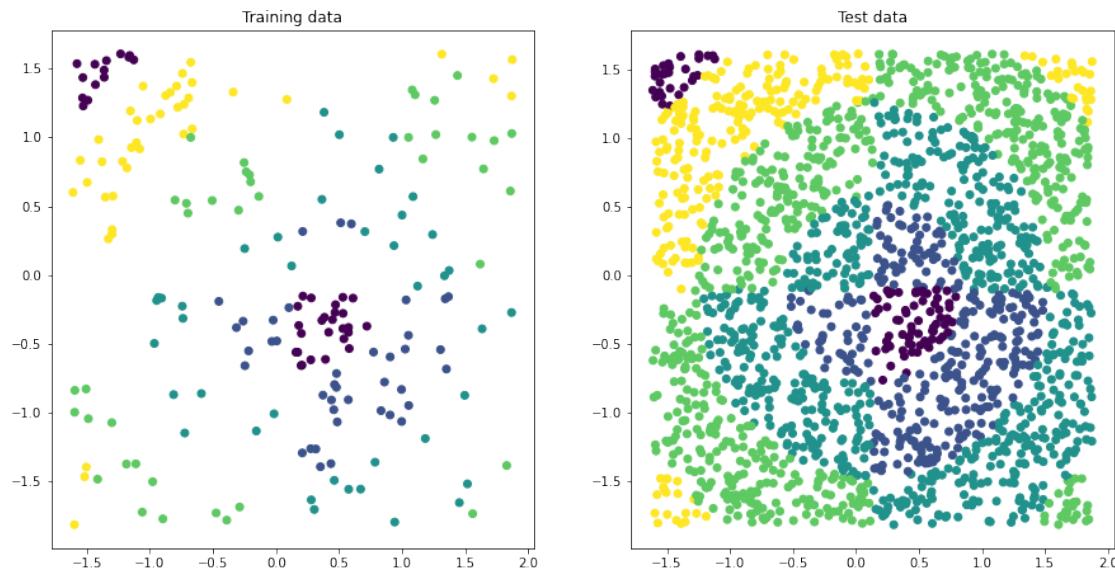
## 21 Eksperyment 6 regularyzacja i walidacja

Sprawdzamy jak działa mechanizm regularyzacji i walidacji. Za zbiór walidacyjny będziemy brać zbiór testowy. Walidacja będzie działać w ten sposób że jak k razy pod rząd pogorszy się na walidacyjnym to zaprzestajemy dalszej nauki. Regularyzacja jest L1 i L2 z alpha (współczynnikiem regularyzacji) równym 0.01. Funkcją aktywacji będzie tanh, architekturą [8,8], beta=0.9 i 10% batch. Badane k jest od 0 do 4 z pominięciem 1.

**22 Hipoteza: regularyzacja pogorszy wyniki, ale zmniejszy wagę w sieci**

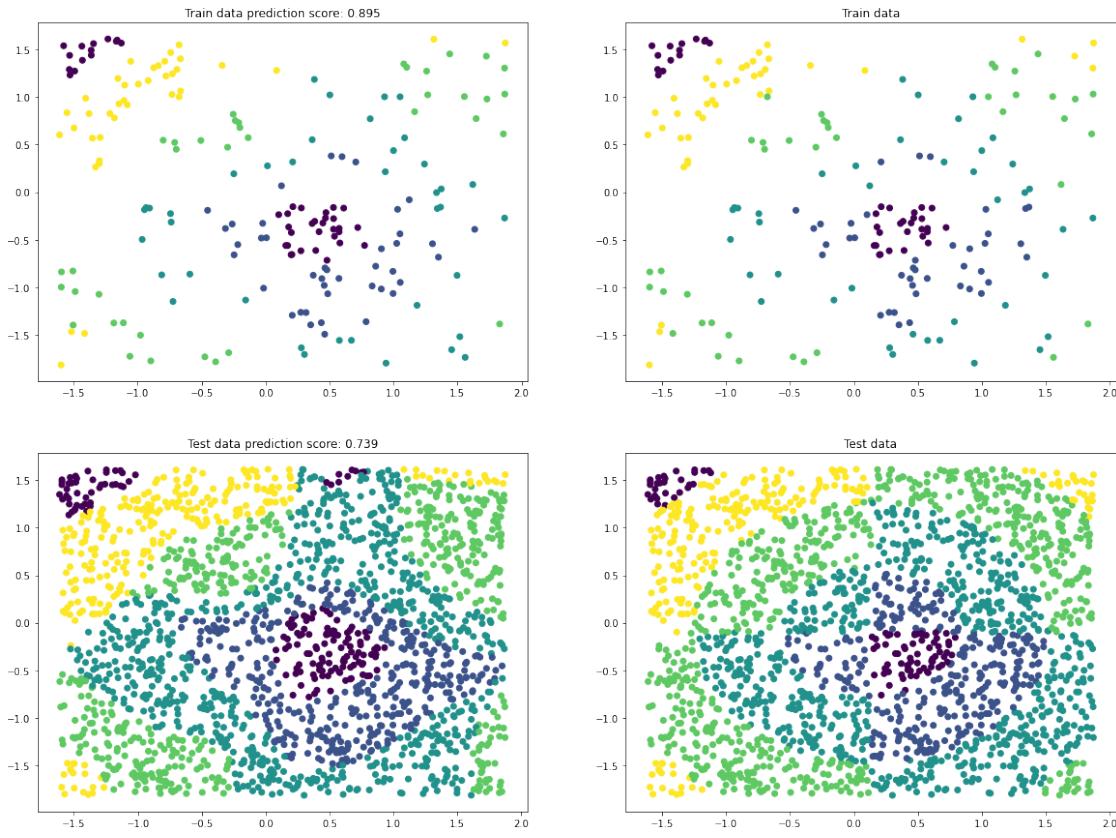
**23 rings5-sparse**

[1181]: [Text(0.5, 1.0, 'Test data')]



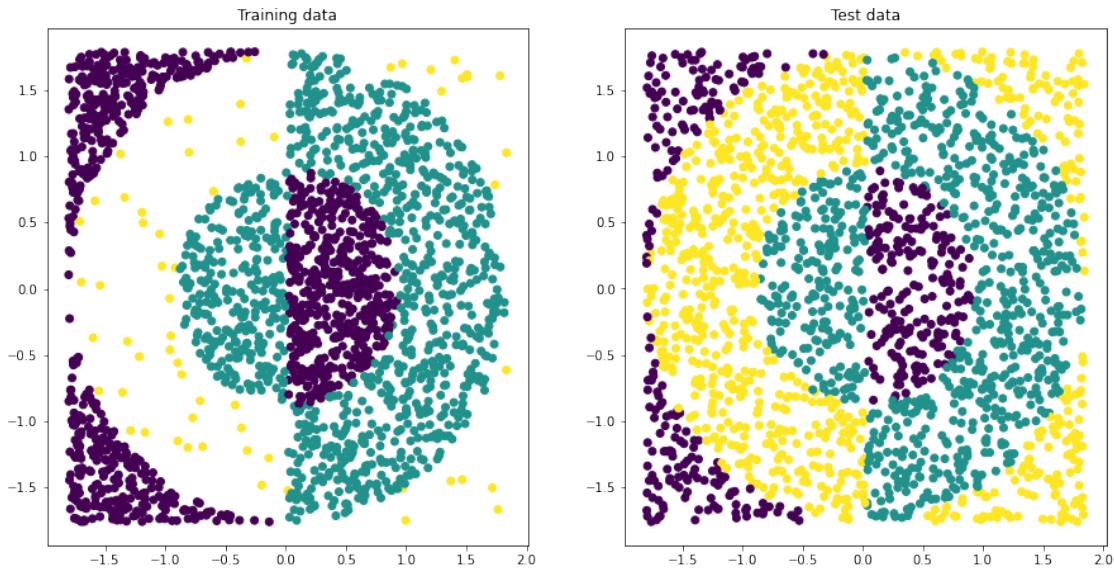
**23.1 Najlepszy model**

[1183]: [Text(0.5, 1.0, 'Test data')]



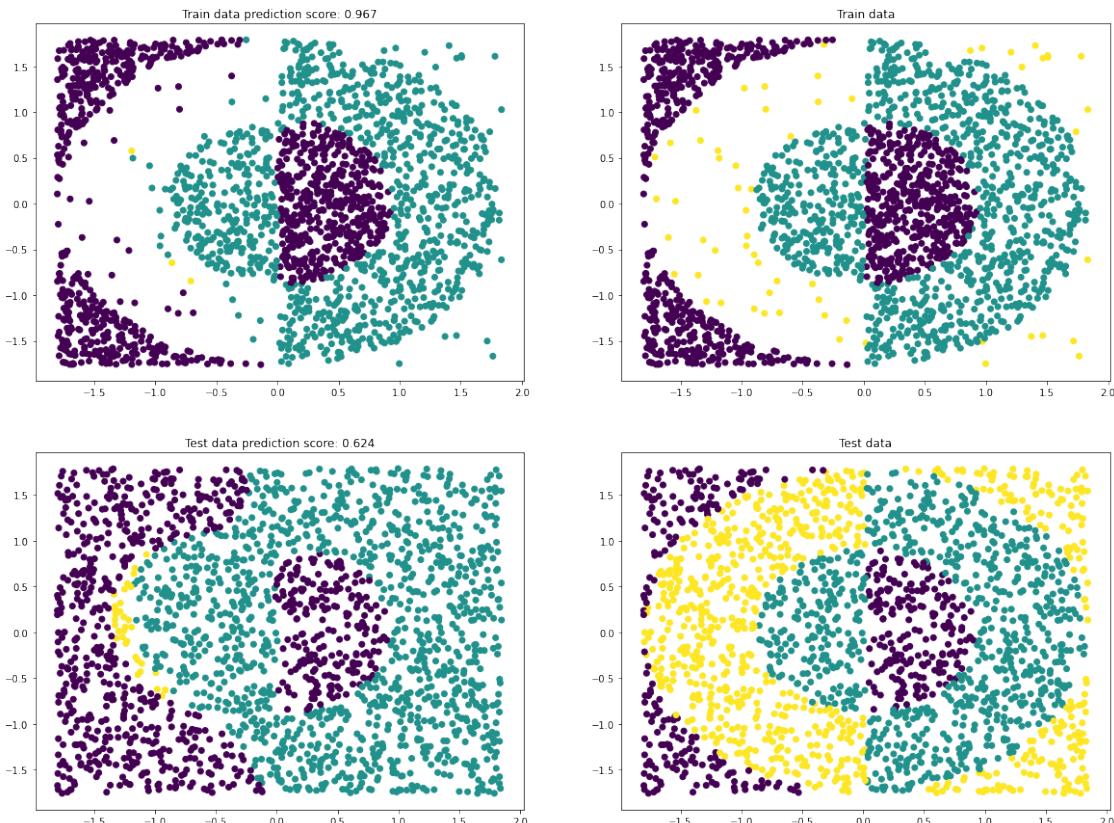
## 24 rings3-balance

```
[1186]: [Text(0.5, 1.0, 'Test data')]
```



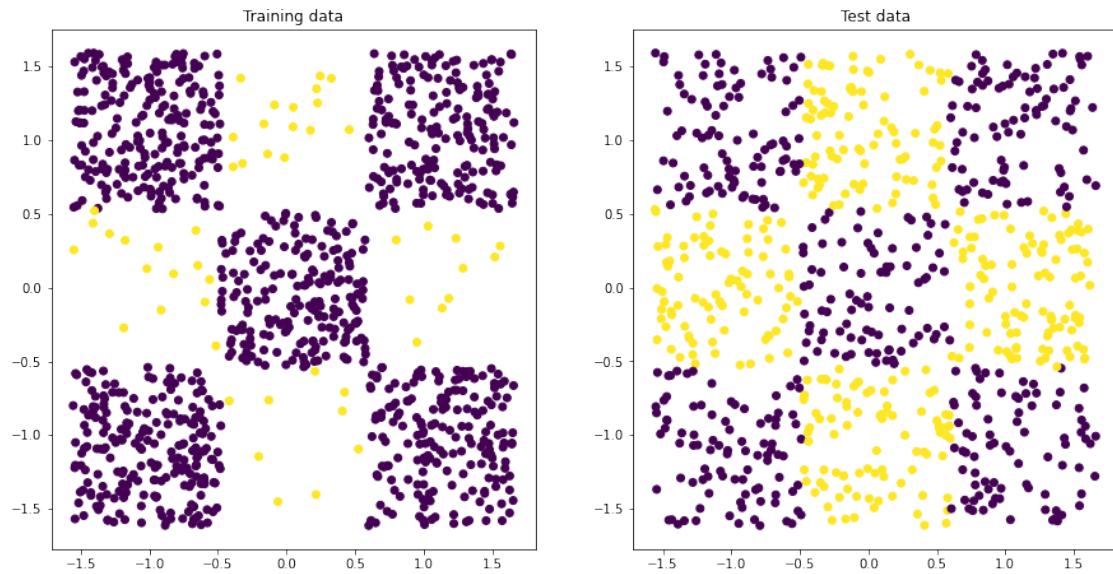
## 24.1 Najlepszy model

```
[1188]: [Text(0.5, 1.0, 'Test data')]
```



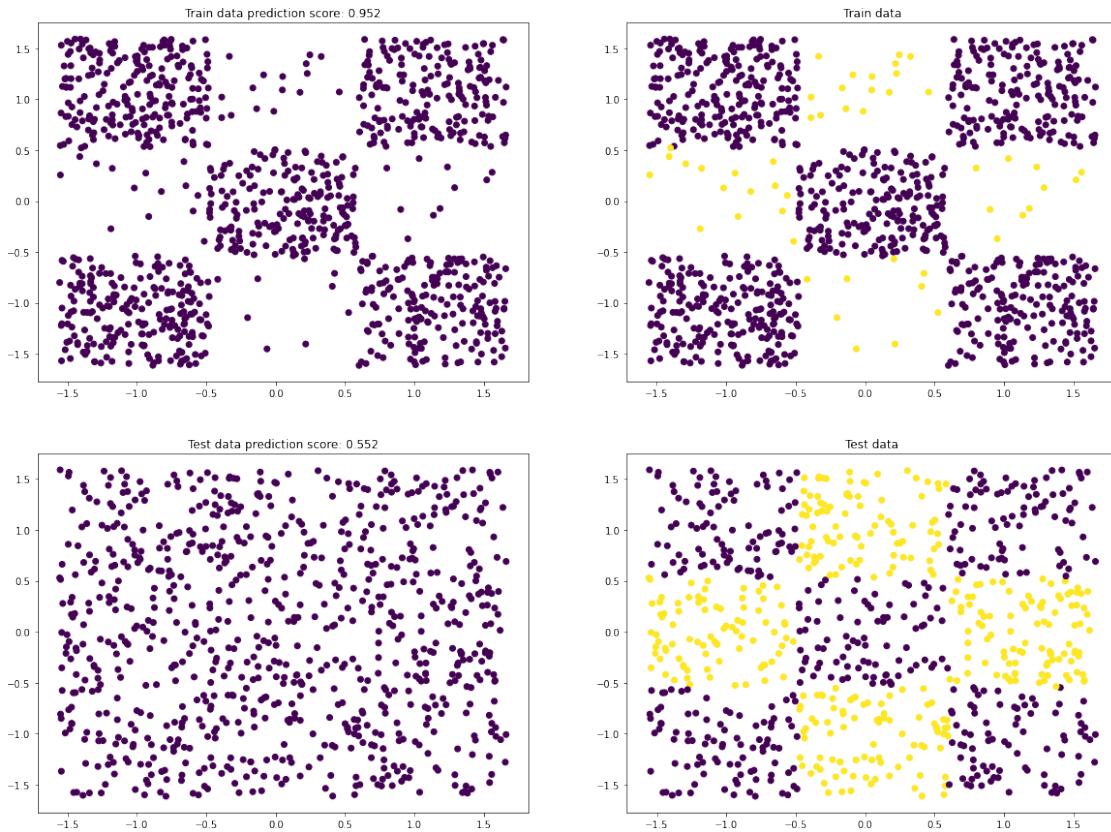
## 25 xor3-balance

```
[1191]: [Text(0.5, 1.0, 'Test data')]
```



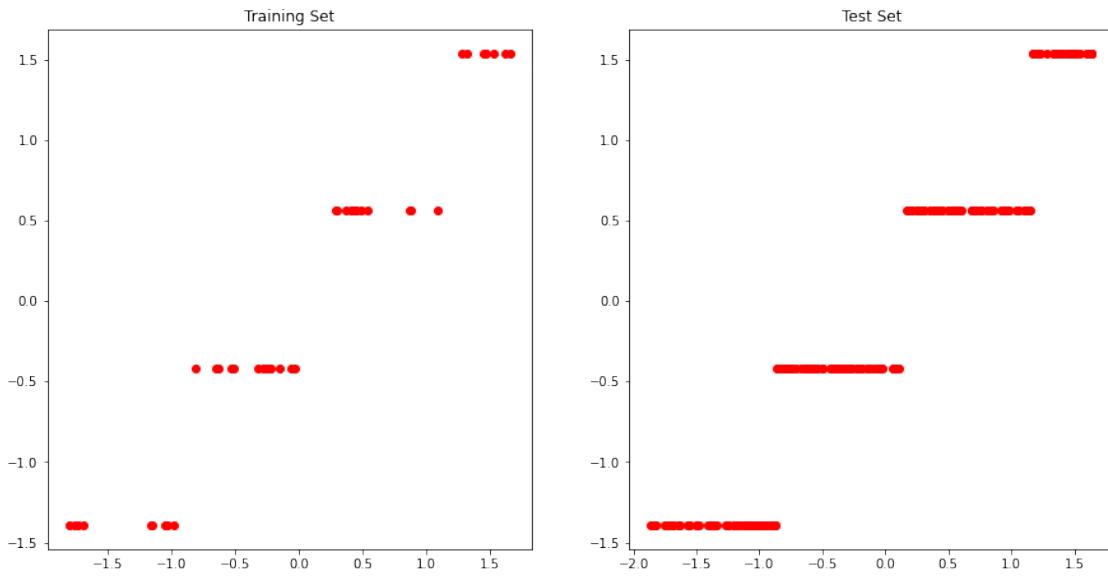
### 25.1 Najlepszy model

```
[1193]: [Text(0.5, 1.0, 'Test data')]
```

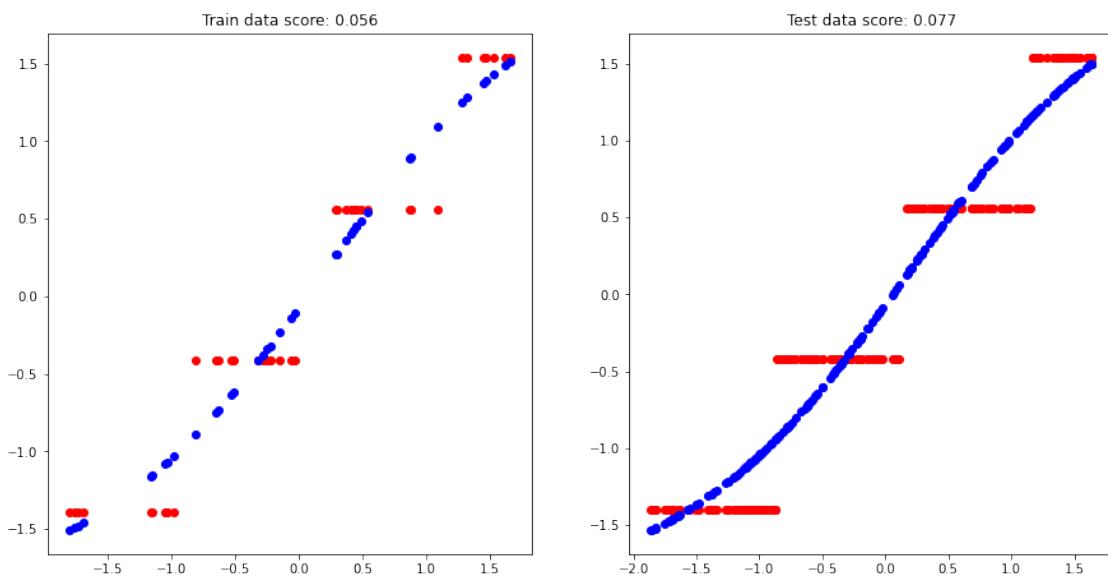


## 26 multimodal-sparse

```
[1196]: [Text(0.5, 1.0, 'Test Set')]
```



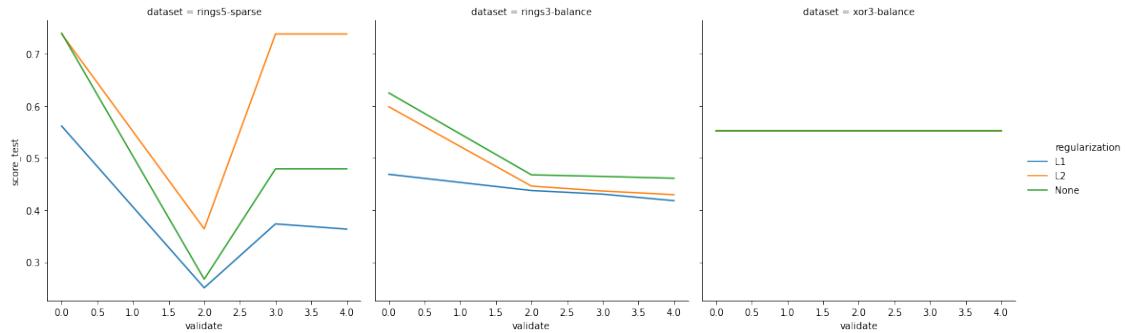
[1198]: [Text(0.5, 1.0, 'Test data score: 0.077')]



## 26.1 Wyniki w zależności od walidacji i regularyzacji

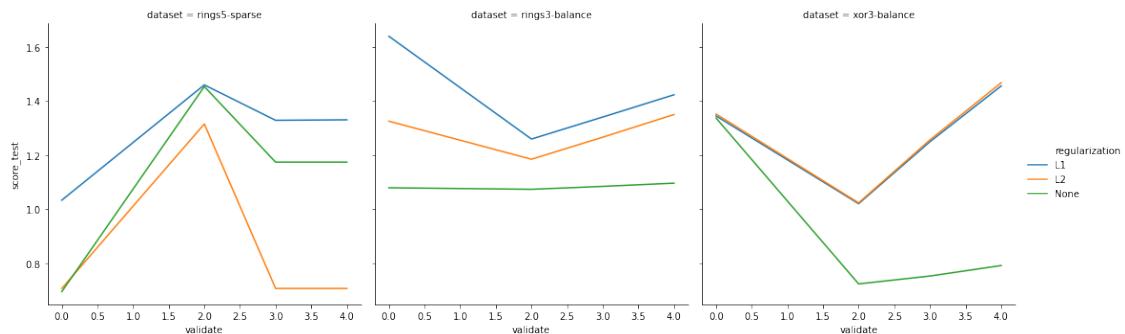
### 26.2 Accuracy

[1203]: <seaborn.axisgrid.FacetGrid at 0x7f9eb36432b0>



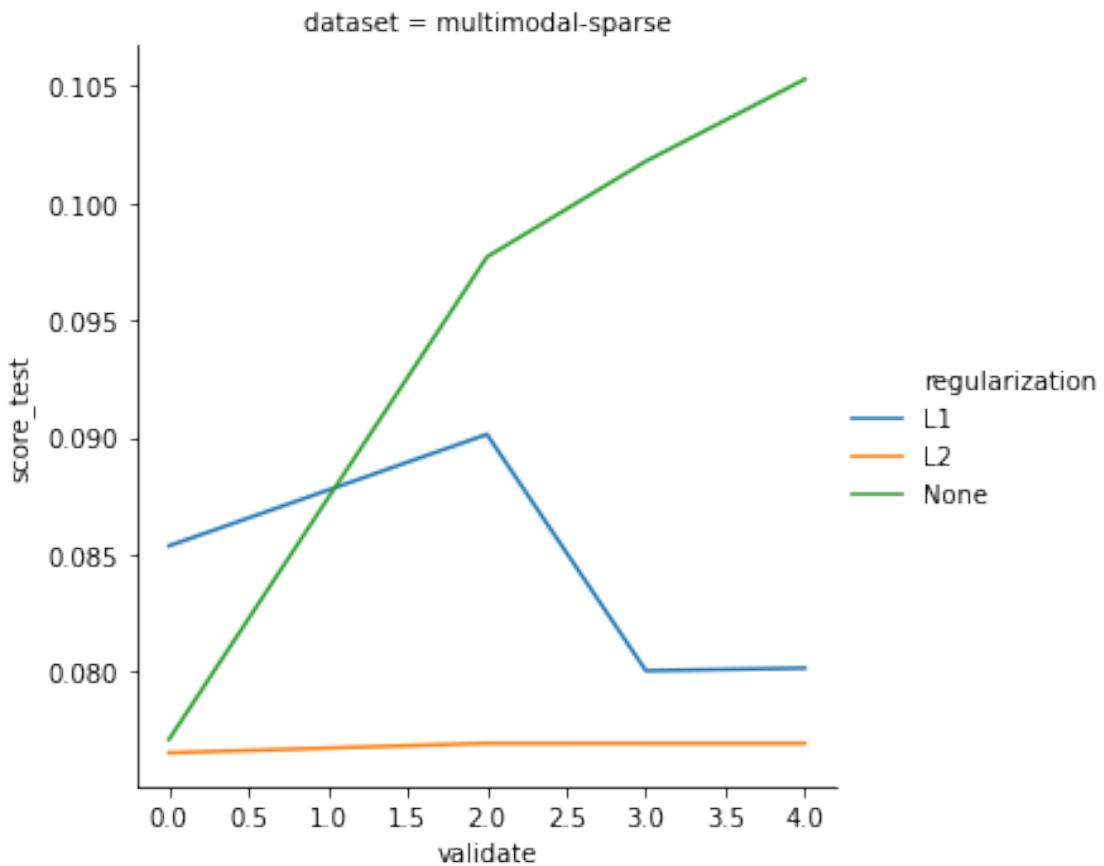
### 26.3 Entropy

[1204]: <seaborn.axisgrid.FacetGrid at 0x7f9eb3c5cf28>



### 26.4 MSE

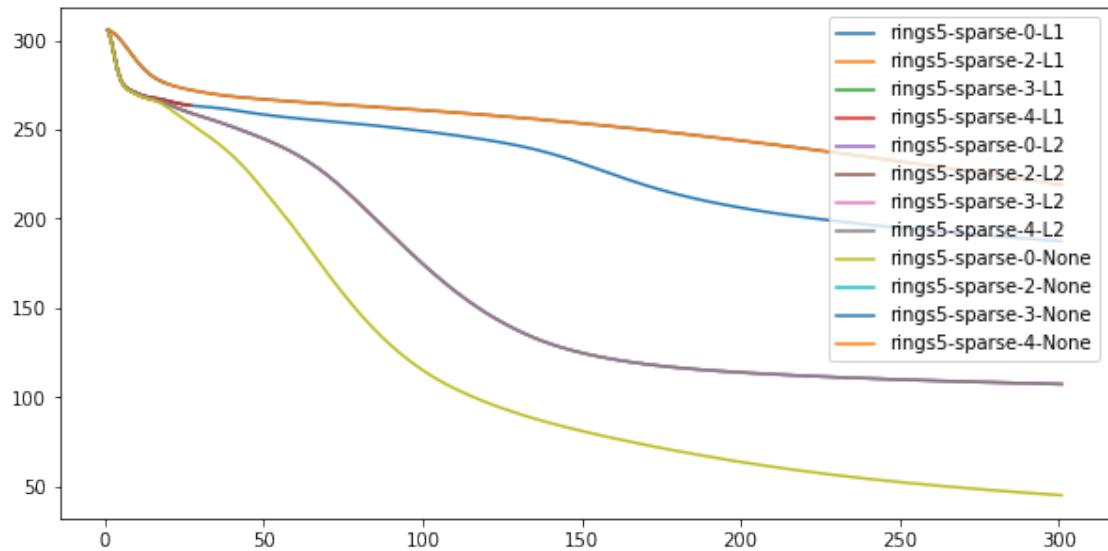
[1205]: <seaborn.axisgrid.FacetGrid at 0x7f9eb331fe10>



Zbyt szybkie zatrzymywanie znacząco pogarsza wynik. Zaś czasami L2 polepszała wynik, a raz pogarszała.

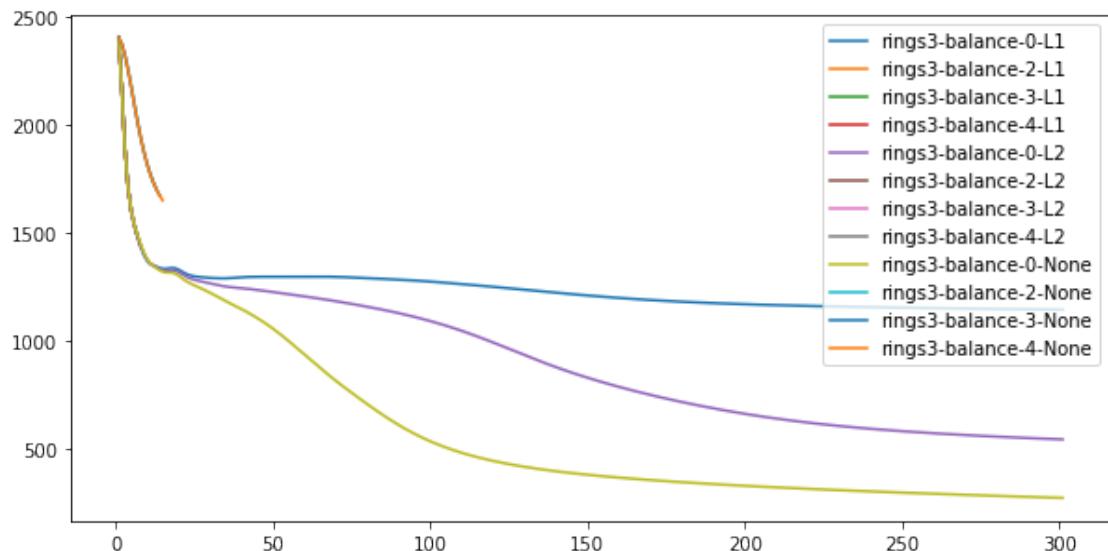
## 26.5 Wykres błędów ring5-sparse

[1207]: <matplotlib.legend.Legend at 0x7f9eb32422b0>



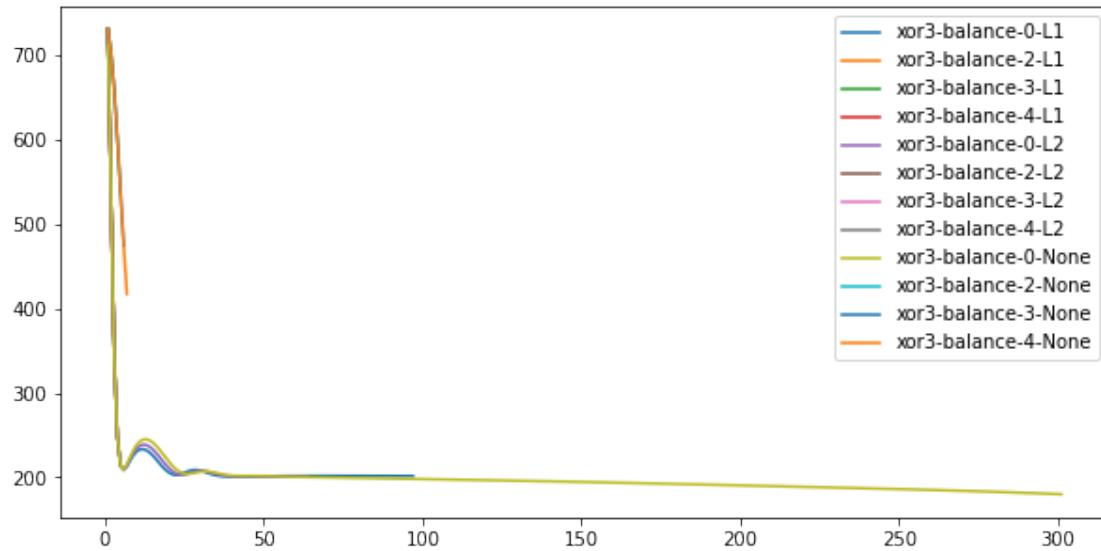
## 26.6 Wykres błędów ring3-balance

[1208]: <matplotlib.legend.Legend at 0x7f9eb31f27f0>



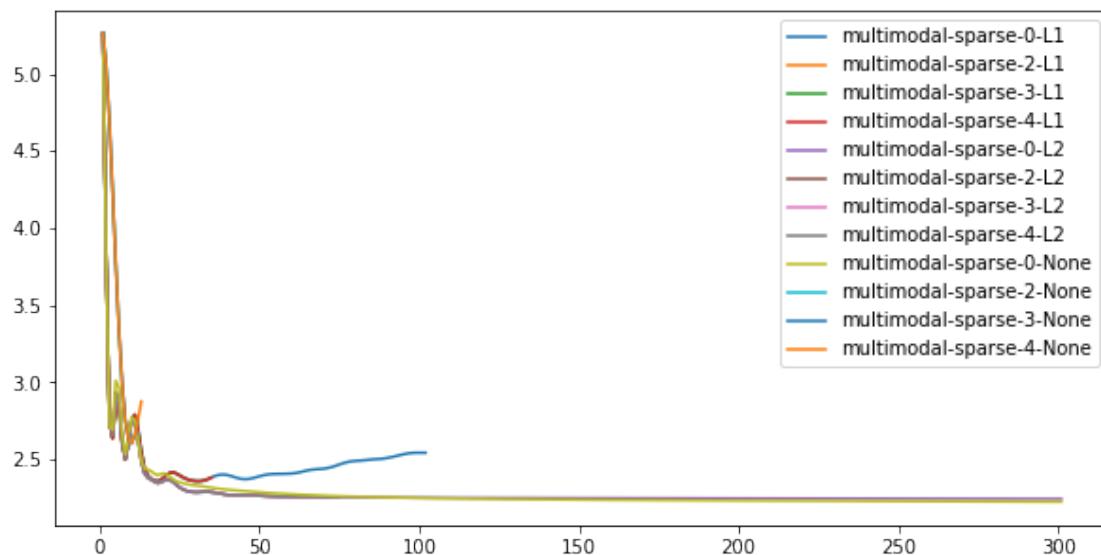
## 26.7 Wykres błędów xor3-balance

[1209]: <matplotlib.legend.Legend at 0x7f9eb312b2e8>



## 26.8 Wykres błędów multimodal-sparse

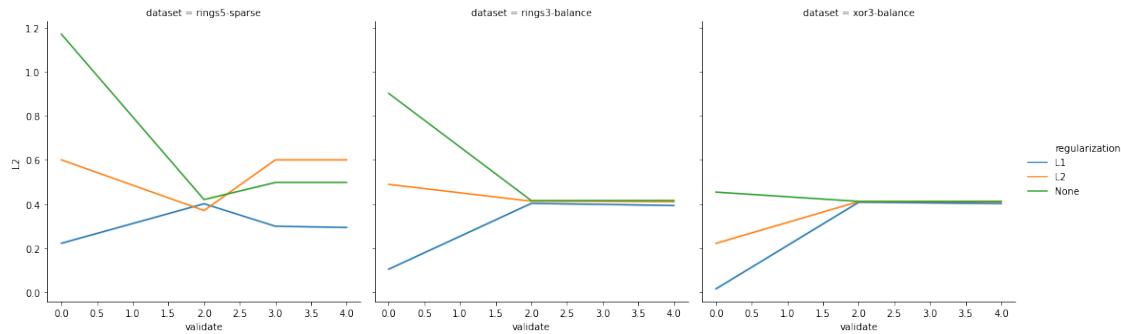
[1210]: <matplotlib.legend.Legend at 0x7f9eb31d2940>



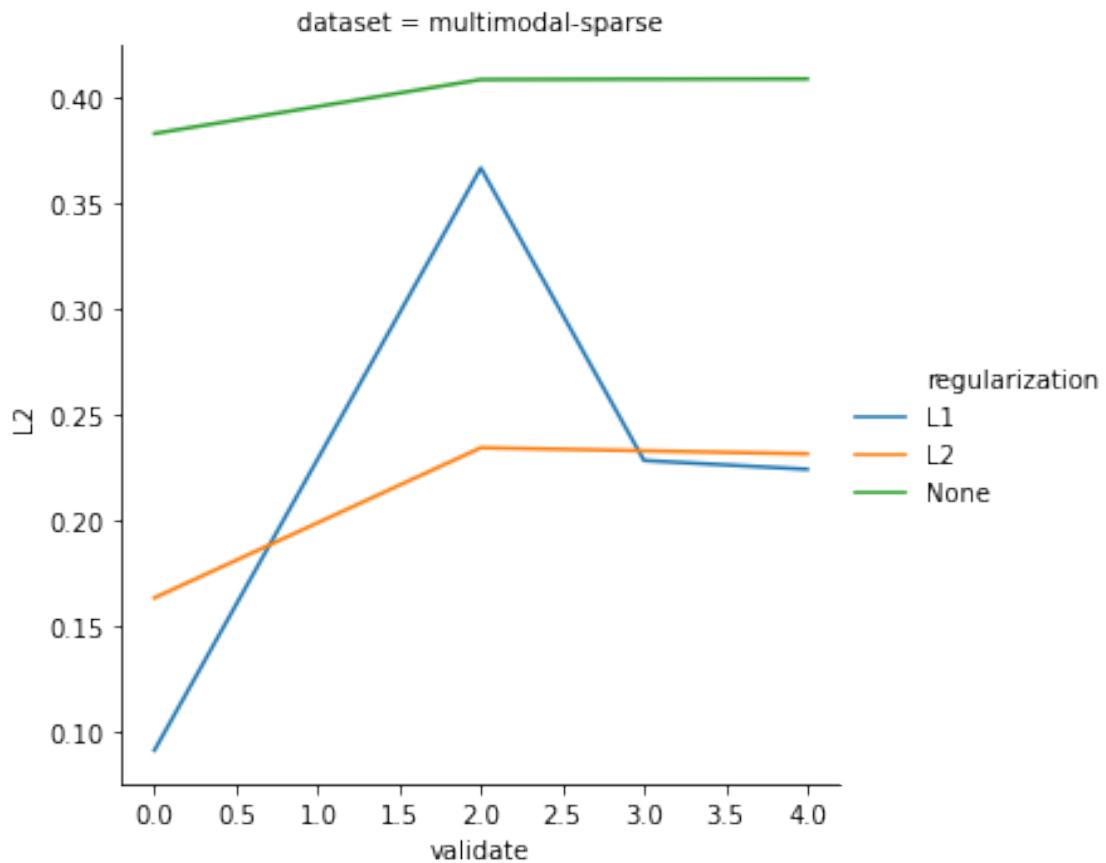
Bez regularyzacji szybciej zbiega

## 26.9 Wykres L2 od regularyzacji i validacji

[1211]: <seaborn.axisgrid.FacetGrid at 0x7f9eb327d828>

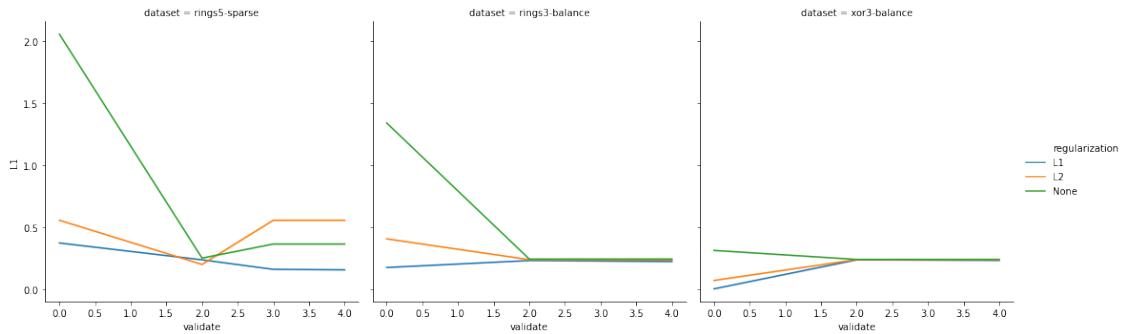


[1212]: <seaborn.axisgrid.FacetGrid at 0x7f9eb30b3be0>

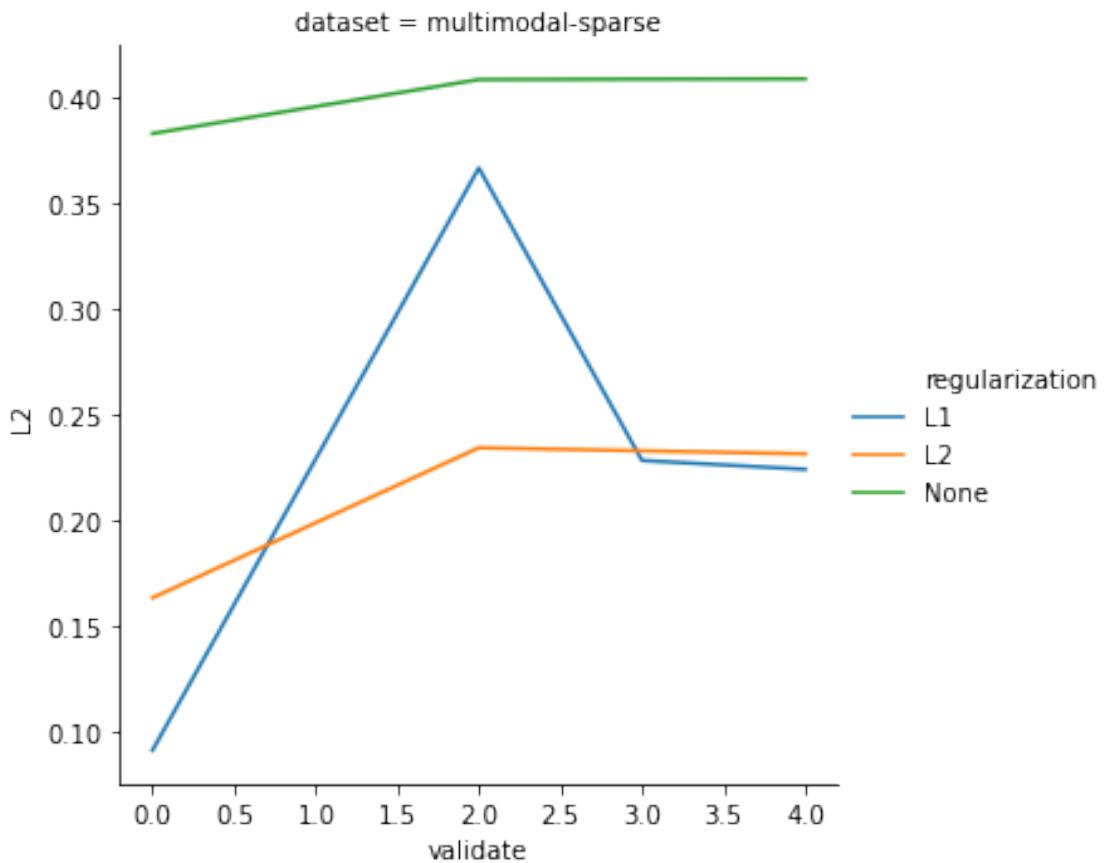


## 26.10 Wykres L1 od regularyzacji i validacji

[1213]: <seaborn.axisgrid.FacetGrid at 0x7f9eb2ec6470>



[1214]: <seaborn.axisgrid.FacetGrid at 0x7f9eb2c59f98>



## **27 Podsumowanie**

Regularyzacja działa w sensie zmniejsza wagi, przy czym L1 regularyzacja przy tej samej alphie bardziej L2 regularyzuje niż nawet L2.